

TEMA 1. Clases y objetos

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

Respuesta

Las cuatro características fundamentales de la POO son: **abstracción, encapsulación, herencia y polimorfismo**.

La **abstracción** permite representar objetos del mundo real mediante clases que capturan sus características esenciales, ignorando los detalles innecesarios. Por ejemplo, una clase `Coche` puede tener atributos como velocidad y color sin necesidad de representar cada átomo del vehículo.

La **encapsulación** consiste en agrupar los datos (atributos) y las operaciones (métodos) que actúan sobre ellos dentro de una clase, ocultando los detalles internos del funcionamiento. Esto es similar a tener funciones y variables relacionadas juntas en C, pero con la capacidad de controlar quién puede acceder a qué mediante modificadores de visibilidad.

La **herencia** permite que una clase herede atributos y métodos de otra clase base, evitando duplicación de código. Una clase `vehículo` podría tener subclases como `Coche` y `Moto` que heredan sus características comunes.

El **polimorfismo** permite que objetos de diferentes clases se comportan de formas distintas ante el mismo mensaje o llamada a método. Por ejemplo, un método `acelerar()` puede comportarse diferente en un `Coche` que en un `Barco`.

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

Respuesta

Cuatro lenguajes populares con soporte para POO son: **Java, C++, Python y C#**. Java es utilizado extensamente en aplicaciones empresariales; C++ añade características orientadas a objetos al C

tradicional que ya se conoce; Python es un lenguaje interpretado de propósito general muy utilizado en ciencia de datos; C# es similar a Java pero diseñado por Microsoft para la plataforma .NET.

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 utiliza estructuras de control como secuencias, condicionales (if-else) y bucles para organizar el flujo del programa, evitando saltos no controlados (goto). El código C que ya se conoce es un ejemplo de programación estructurada, donde los programas se escriben como una serie de instrucciones que se ejecutan en orden con ramificaciones lógicas.

La **programación modular** lleva la estructura un paso más allá al dividir el programa en módulos independientes (típicamente funciones), cada uno responsable de una tarea específica. En C, esto se consigue agrupando funciones relacionadas en archivos .c y .h separados. La modularidad mejora la reutilización de código y facilita el mantenimiento.

Ambos paradigmas siguen siendo válidos y se utilizan hoy en día, pero la POO introduce un nivel adicional de organización al agrupar datos y funciones relacionadas en objetos, lo que aumenta aún más la modularidad y reduce el acoplamiento entre componentes del programa.

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

Respuesta

Tres elementos definen a un objeto: **identidad, estado y comportamiento**.

La **identidad** es lo que distingue un objeto de otro. Cada objeto es único y ocupa una posición diferente en la memoria, incluso si dos objetos tienen exactamente los mismos atributos. Por ejemplo, dos objetos `Punto` con coordenadas (3, 4) son objetos diferentes aunque sus valores sean idénticos.

El **estado** es el conjunto de valores de los atributos del objeto en un momento determinado. En un objeto `Coche`, el estado podría incluir su velocidad actual, gasolina disponible y color.

El **comportamiento** es lo que el objeto puede hacer, definido por sus métodos. Un coche tiene comportamientos como acelerar, frenar o girar. Estos tres elementos trabajan conjuntamente: los métodos modifican el estado del objeto, y su identidad permanece constante a lo largo de su ciclo de vida.

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** es un molde o plantilla que define la estructura y el comportamiento de un tipo de objeto. Es similar a un `struct` en C, pero con la capacidad adicional de contener métodos (funciones) asociados. Una clase define qué atributos tendrá un objeto y qué operaciones puede realizar.

Un **objeto** es una instancia concreta de una clase: es un ente que existe en memoria con valores específicos para los atributos definidos en la clase. Si una clase es un plano de una casa, un objeto es una casa real construida según ese plano.

Una **instancia** es exactamente lo mismo que un objeto: es una copia concreta de la clase. Se usa el término "instancia" para enfatizar la relación entre el objeto y la clase de la que procede.

No todos los lenguajes orientados a objetos utilizan clases explícitamente. Algunos lenguajes como JavaScript utilizan **prototipos** en lugar de clases, donde los objetos heredan directamente de otros objetos sin necesidad de una clase intermedia. Sin embargo, Java, C++ y C# sí utilizan el concepto de clase.

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

Respuesta

En Java, los objetos se almacenan en el **heap** (montículo), que es una región de memoria dinámica. Esto permite que los objetos tengan un ciclo de vida más flexible, ya que pueden existir mientras haya referencias que los utilicen. En cambio, las variables primitivas, como enteros o caracteres, suelen almacenarse en la pila (stack), que es una memoria más limitada pero más rápida.

En otros lenguajes como C y C++, los objetos también pueden almacenarse en el heap, pero la gestión de la memoria es manual. En C, se utiliza `malloc()` para reservar memoria y `free()` para liberarla, mientras que en C++ se emplean `new` y `delete`. Sin embargo, en Java, la memoria se gestiona automáticamente mediante un mecanismo llamado **recolección de basura**.

La **recolección de basura** es un proceso que se ejecuta en segundo plano durante la ejecución del programa. Su función es identificar y liberar la memoria ocupada por objetos que ya no son accesibles, es decir, aquellos que no tienen referencias activas. Esto evita problemas comunes en C/C++, como las fugas de memoria, pero puede introducir una ligera sobrecarga en el rendimiento del programa cuando el recolector de basura se activa. Algunos lenguajes modernos, como Rust, han optado por no usar recolección de basura, implementando en su lugar un sistema de gestión de memoria más seguro y eficiente.

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

Respuesta

Un **método** es una función que pertenece a una clase y opera sobre los datos (atributos) de un objeto. Los métodos son el equivalente a las funciones que se creaban en C, pero asociadas a una estructura de datos específica (la clase). Los métodos tienen acceso directo a los atributos del objeto sin necesidad de pasarlos como parámetros.

La **sobrecarga de métodos** permite definir varios métodos con el mismo nombre en una clase, siempre que tengan diferentes firmas (diferentes parámetros en tipo o cantidad). Por ejemplo, se podría tener un método `suma(int a, int b)` y otro `suma(double a, double b)`. Cuando se llama al método, Java determina automáticamente cuál ejecutar según el tipo y número de argumentos proporcionados.

Esto proporciona una interfaz más intuitiva para el usuario de la clase, ya que puede llamar a métodos que tengan el mismo nombre para operaciones similares sin importar los tipos de datos específicos. En C, esto se lograría creando funciones con nombres diferentes como `suma_int()` y `suma_double()`, lo que es más tedioso.

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

Aquí se presenta la clase `Punto` con sus atributos y método:

```
public class Punto {  
    double x;  
    double y;  
  
    public double calculaDistanciaAOrigen() {  
        return Math.sqrt(x * x + y * y);  
    }  
}
```

Para utilizar esta clase, se crea una instancia y se invoca el método:

```
public class Main {  
    public static void main(String[] args) {  
        Punto p = new Punto();  
        p.x = 3.0;  
        p.y = 4.0;  
  
        double distancia = p.calculaDistanciaAOrigen();  
        System.out.println("Distancia al origen: " + distancia);  
    }  
}
```

En este ejemplo, se crea un objeto `Punto` llamado `p`, se asignan valores a sus atributos `x` e `y`, y luego se invoca el método `calculaDistanciaAOrigen()` que devuelve 5.0 (la distancia del punto (3, 4) al origen (0, 0)). La palabra clave `new` crea la instancia en el heap, similar a `malloc()` en C.

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

El punto de entrada en un programa Java es el método `main` , que debe tener la firma `public static void main(String[] args)` . Este método es el primero en ejecutarse cuando se inicia un programa, y su función es coordinar la ejecución del resto del código. En Java, todo el código debe estar contenido dentro de una clase, por lo que el método `main` también forma parte de una clase.

La palabra clave `static` indica que un método o atributo pertenece a la clase en sí misma, no a una instancia específica de la clase. Esto significa que no es necesario crear un objeto de la clase para acceder a un método o atributo estático. Por ejemplo, el método `main` es estático porque debe ejecutarse antes de que se creen instancias de la clase. Sin embargo, `static` no se limita al método `main` ; también se utiliza para definir variables o métodos que son compartidos por todas las instancias de una clase. Es importante tener en cuenta que dentro de un método estático no se puede acceder directamente a atributos o métodos no estáticos, ya que estos últimos pertenecen a instancias específicas.

Cuando se combina `static` con `final` , se crea una constante de clase. Esto significa que el valor de la variable no puede cambiar una vez que ha sido inicializado y que es compartido por todas las instancias de la clase. Por ejemplo, `public static final double PI = 3.14159;` define una constante que puede ser utilizada por todas las instancias de la clase sin posibilidad de ser modificada.

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

Respuesta

Para compilar un programa en Java desde la línea de comandos, se utiliza el comando `javac` seguido del nombre del archivo que contiene el código fuente. Por ejemplo, si el archivo se llama `MiPrograma.java` , se debe ejecutar:

```
javac MiPrograma.java
```

Esto generará un archivo con extensión `.class`, que contiene el **byte-code** del programa. Para ejecutar el programa, se utiliza el comando `java` seguido del nombre de la clase principal (sin la extensión `.class`):

```
java MiPrograma
```

Java es un lenguaje **compilado e interpretado**. El código fuente se compila primero en un formato intermedio llamado **byte-code**, que no es directamente ejecutable por el procesador. Este byte-code es independiente de la plataforma, lo que significa que puede ejecutarse en cualquier sistema operativo que tenga una **máquina virtual de Java** (JVM) instalada.

La **máquina virtual de Java** (JVM) es un programa que interpreta el byte-code y lo traduce a instrucciones específicas del sistema operativo y del hardware en tiempo de ejecución. Esto permite que un programa Java sea altamente portátil, ya que el mismo archivo `.class` puede ejecutarse en diferentes plataformas sin necesidad de ser recompilado. Este enfoque se conoce como "escribir una vez, ejecutar en cualquier lugar" (*write once, run anywhere*).

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` en Java tiene dos funciones principales: reserva espacio en el **heap** para un nuevo objeto e invoca el **constructor** de la clase correspondiente. Es similar a la función `malloc()` en C, pero con la ventaja de que también inicializa automáticamente los atributos del objeto.

Un **constructor** es un método especial que se ejecuta automáticamente al crear una instancia de una clase. Su propósito principal es inicializar los atributos del objeto con valores iniciales. El constructor tiene el mismo nombre que la clase y no tiene un tipo de retorno. Si no se define un constructor explícito, Java proporciona uno por defecto que no realiza ninguna inicialización específica.

A continuación, se muestra un ejemplo de una clase `Empleado` con un constructor que inicializa los atributos `DNI`, `nombre` y `apellidos`:

```

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

    public Empleado(String dni, String nombre, String apellidos) {
        this.dni = dni;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }
}

// Uso:
Empleado emp = new Empleado("12345678X", "Juan", "Pérez");

```

En este ejemplo, el constructor recibe tres parámetros y los utiliza para inicializar los atributos del objeto. La palabra clave `this` se emplea para diferenciar entre los nombres de los parámetros y los nombres de los atributos de la clase.

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` en Java se utiliza para referirse al objeto actual dentro de un método o constructor. Es especialmente útil para diferenciar entre los atributos de la clase y los parámetros del método o constructor que tienen el mismo nombre. En lenguajes como C++, también existe una referencia similar llamada `this`, mientras que en Python se utiliza `self` para cumplir la misma función.

En Java, `this` no está disponible en métodos estáticos, ya que estos no están asociados a ninguna instancia específica de la clase. Además, `this` se utiliza implícitamente en la mayoría de los casos, pero puede ser explícito cuando se necesita mayor claridad o para evitar ambigüedades.

Un ejemplo del uso de `this` en la clase `Punto` sería el siguiente:


```
public class Punto {  
    double x;  
    double y;  
  
    public Punto(double x, double y) {  
        this.x = x; // Diferencia entre el atributo x y el parámetro x  
        this.y = y;  
    }  
  
    public double calculaDistanciaAOrigen() {  
        return Math.sqrt(this.x * this.x + this.y * this.y);  
    }  
}
```

En este ejemplo, `this` se utiliza en el constructor para diferenciar entre los parámetros `x` e `y` y los atributos de la clase con el mismo nombre. Aunque en el método `calculaDistanciaAOrigen()` el uso de `this` es opcional, puede ser útil para hacer explícito que se está accediendo a los atributos del objeto actual.

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

Aquí se muestra el método `distanciaA` añadido a la clase `Punto`:

```

public class Punto {
    double x;
    double y;

    public Punto(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double calculaDistanciaAOrigen() {
        return Math.sqrt(this.x * this.x + this.y * this.y);
    }

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

// Uso:
Punto p1 = new Punto(0, 0);
Punto p2 = new Punto(3, 4);
double dist = p1.distanciaA(p2); // Devuelve 5.0

```

El método `distanciaA` recibe un objeto `Punto` como parámetro, accede a sus atributos públicos `x` e `y`, y calcula la distancia euclidiana entre el punto actual (`this`) y el punto proporcionado. Este ejemplo demuestra cómo los métodos pueden operar sobre múltiples objetos de la misma clase.

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, los objetos como `Punto` se pasan **por referencia**: se pasa una referencia (similar a un puntero en C) al objeto original. Si se modifica un atributo del objeto dentro del método, dichos cambios afectan al objeto original fuera del método. Esto es diferente a C/C++, donde se pueden pasar referencias explícitamente con `&` o usar punteros.

Los tipos primitivos como `int`, `double`, `boolean`, etc., se pasan **por copia**: se crea una copia del valor, y cualquier modificación dentro del método no afecta la variable original. Esto es similar al paso por valor en C/C++.

```
public void modificaPunto(Punto p) {
    p.x = 100; // Afecta al objeto original
}

public void modificaEntero(int valor) {
    valor = 100; // No afecta a la variable original
}

// Uso:
Punto p = new Punto(5, 5);
modificaPunto(p);
// Ahora p.x es 100

int num = 5;
modificaEntero(num);
// num sigue siendo 5
```

Esta distinción es fundamental: los objetos son referencias (paso por referencia), mientras que los primitivos son valores (paso por copia).

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` en Java que devuelve una representación textual del objeto. Cuando se imprime un objeto usando `System.out.println()`, Java automáticamente invoca `toString()` para convertir el objeto en una cadena de texto. Por defecto, devuelve algo como `Punto@1a2b3c`, que no es muy útil.

El concepto existe en otros lenguajes con nombres diferentes. En C++, se sobrecarga el operador `<<` o se define un método similar. En Python, se sobrecarga el método `__str__()`. La idea es la misma: proporcionar una forma legible de representar el estado del objeto como texto.

Aquí se muestra un ejemplo de `toString()` sobrecargado en la clase `Punto`:

```
public class Punto {
    double x;
    double y;

    public Punto(double x, double y) {
        this.x = x;
        this.y = y;
    }

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

// Uso:
Punto p = new Punto(3, 4);
System.out.println(p); // Imprime: Punto(3.0, 4.0)
```

La anotación `@Override` es opcional pero indica explícitamente que se está sobrescribiendo un método de la clase base. Ahora cuando se imprime el objeto, se obtiene una representación clara y legible.

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

Una clase es conceptualmente similar a un `struct` en C en que ambas agrupan datos relacionados en una única entidad. Sin embargo, existen diferencias significativas. Un `struct` en C es principalmente un contenedor pasivo de datos, mientras que una clase incluye comportamiento (métodos).

Para que un `struct` en C fuera equivalente a una clase, le faltaría: **(1) métodos o funciones miembro** que operen sobre los datos; **(2) encapsulación y control de acceso** mediante modificadores de visibilidad como `private` y `public`; y **(3) constructores y destructores** para inicializar y limpiar automáticamente los datos.

En C es posible emular algunas de estas características pasando un puntero a la estructura como primer parámetro a una función (similar a `this`), pero esto es manual y poco elegante. Java proporciona estos mecanismos de forma integrada y automática, lo que hace que las variables de tipo clase sean verdaderas instancias con comportamiento, no solo contenedores de datos. Esto es una de las ventajas fundamentales de la POO sobre la programación estructurada.

17. Quidemos 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

Se puede emular la clase `Punto` de Java usando un `struct` en C y funciones que reciben un puntero al `struct` como primer parámetro:

```

#include <math.h>

typedef struct {
    double x;
    double y;
} Punto;

double Punto_calculaDistanciaAOrigen(Punto *this) {
    return sqrt(this->x * this->x + this->y * this->y);
}

// Uso:
Punto p;
p.x = 3.0;
p.y = 4.0;
double distancia = Punto_calculaDistanciaAOrigen(&p);

```

En este ejemplo, `this` ha sido reemplazado explícitamente por un puntero al `struct` que se pasa como parámetro. El nombre de la función incluye el nombre del struct (`Punto_`) para simular un espacio de nombres, ya que C no tiene soporte nativo para métodos.

La diferencia principal es que en Java, `this` es implícito y automático, mientras que en C debe ser explícitamente pasado como parámetro. Esto ilustra cómo la POO en Java oculta una gran cantidad de detalles de bajo nivel que en C deben gestionarse manualmente. Java hace que el código sea más legible (`p.calculaDistanciaAOrigen()` frente a `Punto_calculaDistanciaAOrigen(&p)`) y proporciona mejor encapsulación.