

TEMA 1. Clases y objetos

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

Respuesta

La primera característica fundamental de la programación orientada a objetos es la abstracción, que consiste en modelar entidades del mundo real atendiendo únicamente a los aspectos relevantes para el problema que se quiere resolver. Se eliminan los detalles innecesarios y se trabaja con conceptos más cercanos al dominio del problema, facilitando así la comprensión del código.

La segunda característica es la encapsulación, que permite agrupar datos y comportamiento relacionados dentro de una misma entidad, ocultando los detalles internos de implementación. Esto evita accesos indebidos al estado interno de los objetos y reduce el acoplamiento entre distintas partes del programa.

La herencia es otra característica clave y permite crear nuevas clases a partir de otras existentes, reutilizando código y estableciendo relaciones jerárquicas. Gracias a la herencia, se pueden definir comportamientos comunes en una clase base y especializarlos en clases derivadas.

Por último, el polimorfismo permite tratar objetos de diferentes clases de forma uniforme a través de una interfaz común. Esto posibilita que una misma operación se comporte de manera distinta según el objeto concreto sobre el que actúe, aumentando la flexibilidad y extensibilidad del software.

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

Respuesta

Uno de los lenguajes orientados a objetos más conocidos es Java, diseñado desde su origen para seguir este paradigma. En Java, todo el código se organiza en clases y objetos, lo que lo convierte en un ejemplo claro de lenguaje orientado a objetos puro.

Otro lenguaje muy popular es C++, que extiende el lenguaje C añadiendo soporte para clases, objetos, herencia y polimorfismo. C++ es un lenguaje multiparadigma, ya que permite tanto programación estructurada como orientada a objetos.

Python también soporta programación orientada a objetos, aunque no obliga a utilizarla. Permite definir clases, crear objetos y usar herencia y polimorfismo, combinándolo con otros estilos de programación de forma flexible.

Por último, C# es un lenguaje moderno desarrollado por Microsoft que sigue un modelo muy similar al de Java. Está fuertemente orientado a objetos y se utiliza ampliamente en aplicaciones de escritorio, web y videojuegos.

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 busca organizar el código utilizando estructuras de control claras como secuencia, selección y repetición. Se evita el uso de saltos incontrolados (como goto) y se promueve un flujo de ejecución más legible y predecible, como ocurre en C mediante if, for o while.

La programación modular va un paso más allá y propone dividir un programa en módulos independientes, cada uno encargado de una parte concreta del problema. En C, estos módulos suelen corresponderse con archivos .c y .h, donde se separa la implementación de la interfaz.

El objetivo principal de la programación modular es mejorar la mantenibilidad y reutilización del código. Cada módulo puede desarrollarse, probarse y modificarse de manera relativamente independiente del resto del sistema.

Aunque estos paradigmas no incluyen conceptos como objetos o clases, sentaron las bases para la programación orientada a objetos al fomentar la organización, claridad y separación de responsabilidades en el software.

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

Respuesta

Un objeto en programación orientada a objetos se define, en primer lugar, por su identidad, que permite distinguirlo de otros objetos incluso si tienen el mismo estado. Dos objetos pueden tener los mismos valores en sus atributos y, aun así, ser entidades distintas en memoria.

El segundo elemento es el estado, que está formado por el conjunto de atributos o variables internas del objeto. Estos atributos representan la información que describe al objeto en un momento determinado, como podrían ser las coordenadas de un punto o el nombre de un empleado.

El tercer elemento es el comportamiento, que corresponde a los métodos u operaciones que el objeto puede realizar. Estos métodos definen cómo el objeto interactúa con otros objetos y cómo puede modificar su propio estado.

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 una plantilla o definición que describe las características y comportamientos que tendrán los objetos creados a partir de ella. Define qué atributos tendrá un objeto y qué métodos podrá ejecutar, pero no representa un objeto concreto en sí misma.

Un objeto es una entidad concreta creada a partir de una clase, mientras que el proceso de crear un objeto a partir de una clase se denomina instanciación. Por tanto, un objeto es una instancia de una clase, del mismo modo que una variable es una instancia de un tipo en C.

No todos los lenguajes orientados a objetos manejan explícitamente el concepto de clase. Existen lenguajes basados en prototipos, como JavaScript, donde los objetos se crean a partir de otros objetos en lugar de a partir de clases tradicionales.

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

Respuesta

La ubicación en memoria de los objetos depende del lenguaje de programación y de su modelo de ejecución. En lenguajes como Java, los objetos se almacenan generalmente en el heap, mientras que las variables que contienen referencias a esos objetos suelen estar en la pila.

No todos los lenguajes gestionan la memoria de la misma forma. En C++, por ejemplo, los objetos pueden crearse tanto en la pila como en el heap, y es responsabilidad del programador liberar la memoria cuando ya no es necesaria.

La recolección de basura es un mecanismo automático de gestión de memoria presente en lenguajes como Java. El sistema detecta qué objetos ya no son accesibles desde el programa y libera automáticamente la memoria que ocupan, evitando fugas de memoria y errores comunes asociados a la gestión manual.

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

Respuesta

Un método es una función asociada a una clase u objeto que define una operación que puede realizarse sobre ese objeto. A diferencia de las funciones en C, los métodos tienen acceso directo a los atributos del objeto al que pertenecen.

La sobrecarga de métodos consiste en definir varios métodos con el mismo nombre dentro de una misma clase, pero con distinta lista de parámetros. El compilador determina cuál de ellos debe ejecutarse en función del número y tipo de argumentos utilizados en la llamada.

Este mecanismo mejora la legibilidad del código, ya que permite usar un mismo nombre para operaciones conceptualmente similares, evitando la proliferación de nombres distintos para acciones relacionadas.

8. Ejemplo mínimo de clase en Java, que se llame Punto, con dos atributos, x e y, con un método que se llame **calculaDistanciaAOri**gen, 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

Ejemplo 1:

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

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

Ejemplo 2:

```
class Punto {  
    double x;  
    double y;  
  
    double calculaDistanciaAOriente() {  
        return Math.sqrt(x * x + y * y);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Punto p = new Punto();  
        p.x = 3;  
        p.y = 4;  
        System.out.println(p.calculaDistanciaAOriente());  
    }  
}
```

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 de un programa en Java es el método main, cuya firma es fija y reconocida por la máquina virtual. Este método es el primero que se ejecuta cuando se lanza un programa Java desde la línea de comandos o un entorno de desarrollo.

La palabra clave static indica que un método o atributo pertenece a la clase y no a una instancia concreta. Esto permite acceder a él sin necesidad de crear un objeto, lo cual es necesario para que la JVM pueda ejecutar main sin instancias previas.

El modificador static no se limita al método main, sino que puede emplearse en otros métodos y atributos cuando se desea que sean compartidos por todas las instancias de una clase. Cuando se combina con final, se utiliza para definir constantes o métodos que no deben ser modificados.

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 compilar un programa Java desde la línea de comandos se utiliza la herramienta javac, que transforma el código fuente .java en archivos .class. Posteriormente, el programa se ejecuta mediante el comando java, que lanza la máquina virtual.

Java es un lenguaje compilado e interpretado. El código fuente se compila a un formato intermedio llamado byte-code, que no es código máquina nativo, sino un conjunto de instrucciones para la máquina virtual de Java.

La máquina virtual (JVM) es un entorno que interpreta o ejecuta el byte-code y lo adapta al sistema operativo subyacente. Esto permite que un mismo programa Java pueda ejecutarse en diferentes plataformas sin recompilar.

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

El operador new se utiliza en Java para crear un nuevo objeto en memoria y devolver una referencia a él. Este operador reserva espacio para el objeto e invoca automáticamente a su constructor.

Un constructor es un método especial que se ejecuta cuando se crea un objeto y sirve para inicializar su estado. Tiene el mismo nombre que la clase y no devuelve ningún valor.

```
class Empleado {  
    String dni;  
    String nombre;  
    String apellidos;  
  
    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 representa al objeto actual sobre el que se está ejecutando un método. Permite acceder explícitamente a los atributos y métodos de la instancia desde dentro de la propia clase.

El uso de this es especialmente útil cuando los parámetros de un método o constructor tienen el mismo nombre que los atributos de la clase, ya que permite evitar ambigüedades.

No todos los lenguajes utilizan el mismo nombre para esta referencia. En C++, por ejemplo, se emplea también `this`, mientras que en otros lenguajes puede recibir nombres distintos o ser implícita.

```
class Punto {  
    double x, y;  
  
    void setX(double x) {  
        this.x = x;  
    }  
}
```

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

```
double distanciaA(Punto otro) {  
    double dx = this.x - otro.x;  
    double dy = this.y - otro.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, los parámetros se pasan siempre por valor, pero en el caso de los objetos lo que se copia es la referencia al objeto, no el objeto en sí. Esto implica que los cambios realizados sobre los atributos del objeto dentro del método sí afectan al objeto original.

Si dentro del método se reasigna la referencia a otro objeto distinto, ese cambio no se refleja fuera del método, ya que solo se modifica la copia local de la referencia.

En el caso de tipos primitivos como `int`, el valor se copia directamente. Por tanto, cualquier modificación realizada dentro del método no afecta a la variable original, comportándose como 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 heredado de la clase `Object` que se utiliza para obtener una representación textual de un objeto. Por defecto devuelve información poco útil, como la dirección de memoria lógica del objeto.

Es habitual sobrescribir este método para proporcionar una representación más significativa del objeto, especialmente útil para depuración o salida por pantalla.

Este concepto existe en otros lenguajes, como C++ mediante la sobrecarga del operador `<<` o funciones equivalentes para convertir objetos a texto.

```
@Override  
public String toString() {  
    return "Punto(" + x + ", " + 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

Una clase puede considerarse conceptualmente similar a un struct en C en cuanto a que ambos agrupan datos relacionados bajo un mismo nombre. Sin embargo, un struct es solo una agrupación de datos, sin comportamiento asociado.

Para que un struct se parezca a una clase, debería incorporar funciones que operen sobre sus datos y algún mecanismo de control de acceso. En C, esto se simula mediante funciones externas que reciben el struct como parámetro.

Además, una clase permite crear instancias con comportamiento propio, herencia y encapsulación, características que no existen de forma nativa en C.

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

```
#include <math.h>  
  
typedef struct {  
    double x;  
    double y;  
} Punto;  
  
double calculaDistanciaAOriente(Punto *p) {  
    return sqrt(p->x * p->x + p->y * p->y);  
}
```

En este enfoque, el struct contiene solo los datos, mientras que la función recibe explícitamente un puntero al struct. La referencia this desaparece y se sustituye por un parámetro explícito que representa al objeto sobre el que se opera.

Este patrón muestra que la programación orientada a objetos puede entenderse como una evolución organizada de técnicas ya existentes en lenguajes procedimentales como C, donde el programador debe gestionar explícitamente aspectos que en lenguajes orientados a objetos están integrados en el propio lenguaje.

Completamos la documentación luego de la clase teórica

(1)

Abstracción Olvidar detalles --> Trata temas complejos y facilita los cambios en el código

Encapsulación Permite unir el estado y el comportamiento

Herencia Establecer jerarquías Ejemplo:

```
Animal {dormir}
```

{ladrar} Perro Gato {maullar}

Animal es la clase base que proporciona comportamientos compartidos como dormir. Las clases Perro y Gato heredan de Animal y añaden comportamientos específicos, por ejemplo ladear y maullar, respectivamente.

Polimorfismo Misma función, distintas formas / o implementaciones

El polimorfismo permite que una misma operación (por ejemplo, dormir) se comporte de diferentes maneras según el tipo concreto del objeto que la ejecuta

Ejemplo:

¿Cómo duerme el perro? ¿Cómo duerme el gato?

(2)

Python & JavaScript --> Dinámicos

Java, C# --> Tienen GC (Recolector de basura) - Compilados - Comprobación estática de tipos

C++ --> No tiene recolector de basura

Recolector de basura --> Componente del runtime que libera automáticamente la memoria ocupada por objetos que ya no son accesibles desde el programa.

Objetivos GC - Evitar fugas de memoria - Evitar errores por liberaciones manuales

(3)

Ensamblador - Secuencia - Salto arbitrario(**GOTO**) --> Salta si resultado != 0

Estructurado / Procedural / Imperativa - Estructuras de control Condicional (if) Iterativa (while, for, do while)

Sin GOTO

Modular - Librerías, paquetes, módulos - Agrupar código para facilitar su uso por otros programas o módulos

(4)

Identidad "equivalente a decir la dirección de memoria" - Cada objeto tiene identidad

Estado --> Atributos, lo que en los Structs eran campos

Structs en C = Una clase en Java

Comportamiento --> Métodos, son las funciones que todos los objetos de esa clase pueden realizar

```
Animal a;           Animal a;  
                  -->  
dormir(a);         a.dormir();
```

(5)

Clase --> Molde para crear instancias durante la ejecución.

Define la estructura del estado y el comportamiento = = atributos métodos

Objeto --> Entidad concreta creada a partir de una clase.