APUNTES TEMA 1: CLASES Y OBJETOS

---

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

1. Abstracción

Es la capacidad de identificar las características y comportamientos esenciales de un objeto, descartando los detalles que no son relevantes para el contexto del problema. Se logra principalmente mediante clases, interfaces y clases abstractas. Por ejemplo, para un sistema de gestión académica, de un "Estudiante" nos interesa su expediente, pero no su color de ojos o su altura.
"Olvidar detalles". Tratar con temas complejos. Facilitar el cambio

2. Encapsulamiento

Consiste en ocultar el estado interno de un objeto y protegerlo de accesos indebidos desde el exterior. Solo se permite la interacción a través de métodos públicos bien definidos. Utilizamos modificadores de acceso (private, protected, public) y los famosos métodos getters y setters. Esto asegura la integridad de los datos. Unir estado y comportamiento. Ocultar partes

3. Herencia

Es el mecanismo por el cual una clase nueva (subclase) adquiere los atributos y métodos de una clase ya existente (superclase). Esto facilita la reutilización de código y la creación de jerarquías. Se implementa con la palabra reservada extends. Por ejemplo, una clase Coche y una clase Moto pueden heredar de una clase superior llamada Vehiculo. Establecer jerarquías

4. Polimorfismo: **(Misma función, distintas formas o implementaciones)**

Es la capacidad que tienen los objetos de responder de diferentes formas a un mismo mensaje o llamada a un método. Permite tratar a objetos de diferentes subclases como si fueran de una clase común. Se manifiesta principalmente a través de la sobrescritura de métodos (@Override). Si Vehiculo tiene un método moverse(), el Coche lo hará rodando y el Avion volando, pero ambos responden a la misma instrucción.

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

1. JAVA
2. C++
3. Python (dinámico)
4. C#

Java, C++, C# son compilados, comprobación estática de tipos

---

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

1. Programación Estructurada:

antes estaba el ensamblador (existía GOTO)

Es un paradigma que se basa en la división del programa en tres estructuras de control básicas:
Secuencia: Las instrucciones se ejecutan una tras otra.
Selección (Decisión): Uso de if, else, switch.
Iteración (Bucles): Uso de while, for, do-while.
Se centra en los procedimientos y funciones (qué debe hacer el programa). El flujo de ejecución es lineal y predecible. (Sin GOTO)

2. Programación Modular:

librerías, paquetes, modulos que agrupan codigo para facilitar su uso por otros programas

Este concepto lleva la programación estructurada un paso más allá para solucionar el problema de los programas gigantescos. Consiste en dividir un programa en partes más pequeñas y manejables llamadas "módulos".
Independencia: Cada módulo tiene una tarea específica (por ejemplo, un módulo para gestionar la base de datos y otro para la interfaz).
Comunicación: Los módulos se comunican entre sí mediante parámetros y valores de retorno.
Ventaja: Permite que varios programadores trabajen en diferentes partes al mismo tiempo sin estorbarse

"Divide y vencerás". Si un módulo falla, es mucho más fácil de localizar y reparar que un código monolítico de miles de líneas.

---

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

1. Estado (Atributos), en los structs eran los campos

El estado representa los datos o características que posee el objeto en un momento determinado. Se define mediante variables (llamadas campos o atributos).
Ejemplo: En un objeto Estudiante, el estado podría ser: nombre = "Juan", edad = 20, matriculado = true.

2. Comportamiento (Métodos), son las funciones que pueden hacer los objetos de esa clase

Es lo que el objeto puede hacer o las acciones que puede realizar. Se define mediante funciones (llamadas métodos). El comportamiento a menudo cambia el estado del objeto.
Ejemplo: El objeto Estudiante puede matricularse(), entregarTarea() o cambiarNombre().

3. Identidad (direccion de memoria)

Es lo que permite distinguir a un objeto de otro, incluso si tienen el mismo estado (los mismos atributos con los mismos valores). En memoria, cada objeto ocupa una dirección única.
Ejemplo: Puedes tener dos objetos Estudiante llamados "Juan" de 20 años, pero son dos personas distintas. En Java, la identidad se gestiona mediante la referencia de memoria que apunta al objeto.

---

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?

- Una clase es el "plano", "molde" o "plantilla" de software. Define qué datos tendrá un objeto (estado) y qué acciones podrá realizar (comportamiento), pero no es el objeto en sí. Es una definición teórica que reside en el código.

- **¿Es lo mismo que un objeto?**

No. La diferencia es de existencia:
La clase es la definición (está en el archivo .java).
El objeto es la realización física de esa definición en la memoria del ordenador mientras el programa se ejecuta. Son variables del tipo de alguna clase con un estado concreto de sus atributos

- **¿Qué es una instancia?**

"Instancia" es simplemente el nombre técnico que le damos al objeto concreto creado a partir de una clase.
El proceso de crear un objeto se llama instanciación.

- **¿Todos los lenguajes POO manejan el concepto de clase?**

No todos. Aunque la gran mayoría (como Java, C++, Python o C#) se denominan "Lenguajes basados en clases", existe otra rama:
POO Basada en Prototipos: El ejemplo más famoso es JavaScript. En estos lenguajes no existen las clases como moldes rígidos; los objetos se crean clonando otros objetos (prototipos) y añadiéndoles propiedades sobre la marcha.

---

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

- **¿Dónde se almacenan los objetos?**

En la mayoría de los lenguajes modernos (como Java), la memoria se divide principalmente en dos zonas: el Stack (Pila) y el Heap (Montículo).
El Heap: Es el lugar donde se almacenan físicamente los objetos. Es un área de memoria grande y dinámica. Cuando haces un new Objeto(), el espacio para sus datos se reserva en el Heap.
El Stack: Aquí se almacenan las referencias (las direcciones de memoria) que apuntan a esos objetos, además de las variables locales y las llamadas a métodos.

- **¿Es igual en todos los lenguajes?**

No. Aquí es donde radica la gran diferencia entre lenguajes de "alto nivel" y de "bajo nivel":
Java, C#, Python: Los objetos van casi siempre al Heap y tú no decides cuándo se borran.
C++: Es el lenguaje de la libertad (y el peligro). Tú puedes elegir:
Crear un objeto en el Stack (se borra automáticamente al salir de la función).
Crear un objeto en el Heap usando new (y si no lo borras manualmente con delete, causas una "fuga de memoria" o memory leak).

- **¿Qué es la Recolección de Basura (Garbage Collection)?**

El Garbage Collector (GC) es un proceso automático de la Máquina Virtual que se encarga de gestionar la memoria por ti.
Su función: Rastrea el Heap buscando objetos que ya no tienen ninguna referencia apuntándoles (es decir, objetos que el programa ya no puede alcanzar ni usar).
Su acción: Cuando identifica un objeto "huérfano", libera el espacio que ocupaba en el Heap para que pueda ser reutilizado.

---

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

Un método es un bloque de código dentro de una clase que define un comportamiento o una acción que el objeto puede realizar. Es el equivalente a las "funciones" en la programación estructurada, pero con una diferencia clave: los métodos tienen acceso a los datos internos (atributos) del objeto que los contiene.
La sobrecarga es la capacidad de definir en una misma clase varios métodos con el mismo nombre, pero con diferente lista de parámetros (diferente número de argumentos o diferentes tipos de datos). No es suficiente cambiar el tipo de retorno solamente

---

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

Implementación de la Clase Punto

```
public class Punto {
  // Atributos con visibilidad por defecto (sin modificador public/private)
  double x;
  double y;

  // Método para calcular la distancia al origen (0,0)
  public double calculaDistanciaAOrigen() {
      return Math.sqrt(Math.pow(this.x, 2) + Math.pow(this.y, 2));
  }
}
```

Ejemplo de Uso (Instanciación)

```
public class PruebaPunto {
  public static void main(String[] args) {
      // 1. Crear la instancia (Objeto)
      Punto miPunto = new Punto();

      // 2. Definir el estado del objeto
      miPunto.x = 3.0;
      miPunto.y = 4.0;

      // 3. Invocar el comportamiento
      double distancia = miPunto.calculaDistanciaAOrigen();

      // 4. Mostrar resultado
      System.out.println("Punto: (" + miPunto.x + ", " + miPunto.y + ")");
      System.out.println("Distancia al origen: " + 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?

- **El Punto de Entrada: public static void main(String[] args)**

En Java, el punto de entrada es el método main. Es el primer lugar donde mira la Máquina Virtual de Java (JVM) para empezar a ejecutar las instrucciones. Si una clase no tiene este método exacto, el programa no puede "arrancar" por sí solo.

public: Permite que la JVM acceda al método desde fuera.
void: Indica que el método no devuelve ningún valor al finalizar.
String[] args: Es un array que permite recibir argumentos desde la terminal al ejecutar el programa.

- **¿Qué es static y para qué sirve?**

La palabra reservada static indica que un miembro (método o atributo) pertenece a la clase en sí, y no a una instancia (objeto) específica.

Utilidad: Permite usar el método o atributo sin necesidad de crear un objeto con new.
En el main: Se usa static porque la JVM necesita ejecutar el programa antes de que exista cualquier objeto en la memoria.

- **¿Solo se emplea para el método main?**

No. El modificador static es muy común en otros contextos:

Atributos estáticos: Variables compartidas por todos los objetos de esa clase. Si un objeto cambia su valor, cambia para todos. (Ejemplo: un contador de cuántos objetos se han creado).

Métodos estáticos: Utilidades que no dependen del estado de un objeto. (Ejemplo: los métodos de la clase Math, como Math.sqrt() o Math.pow()).

Bloques estáticos: Fragmentos de código que se ejecutan una sola vez cuando la clase se carga por primera vez.

- **¿Para qué se combina con final?**

Cuando combinamos static final, estamos creando una Constante Global.

static: Significa que solo existe una copia en memoria para toda la clase.

final: Significa que su valor no puede ser modificado una vez asignado.

---

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?

- **¿Cómo compilar y ejecutar desde la terminal?**

Para trabajar desde la línea de comandos, seguimos dos pasos fundamentales:

Compilación: Transformamos el código humano (.java) en algo que la máquina entienda.

Comando: javac MiPrograma.java

Resultado: Se genera un archivo llamado MiPrograma.class.

Ejecución: Pedimos a la Java Virtual Machine que corra el programa.

Comando: java MiPrograma (Nota: aquí no se pone la extensión .class).

- **¿Java es compilado o interpretado?**

Es ambos. Java utiliza una compilación intermedia. No se compila directamente a código binario de procesador (como C++), ni se interpreta línea a línea desde el código fuente (como el Python antiguo). Se compila a un lenguaje intermedio.

- **¿Qué es el Byte-code y los ficheros .class?**

Cuando ejecutas javac, el compilador genera Byte-code.

Byte-code: Es un conjunto de instrucciones altamente optimizadas que no están diseñadas para un procesador real (Intel, AMD, ARM), sino para un procesador virtual.

Ficheros .class: Son los archivos que contienen este Byte-code. Es el "ejecutable" de Java que puedes llevar de un Windows a un Mac o a un Linux y funcionará igual.

- **¿Qué es la Máquina Virtual de Java (JVM)?**

La JVM es el corazón de la plataforma Java. Es el programa que actúa como un "traductor" en tiempo real.

Su función: Lee el Byte-code de los archivos .class y lo traduce a las instrucciones específicas del procesador donde se esté ejecutando en ese 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

- **¿Qué es new?**

new es un operador cuya única misión es la instanciación. Cuando el ordenador lee new, ocurren tres cosas en milisegundos:

Reserva espacio: Busca un hueco libre en la memoria Heap lo suficientemente grande para guardar el estado del objeto.

Crea la identidad: Genera una dirección de memoria única para ese objeto.

Llama al constructor: Ejecuta el bloque de código que inicializa el objeto.

Sin new, no hay objeto, solo tendrías una variable de referencia vacía (null).

- **¿Qué es un Constructor?**

El constructor es un método especial que se ejecuta automáticamente al crear un objeto. Su función principal es inicializar los atributos para que el objeto empiece su vida con datos válidos.

```
public class Empleado {
// Atributos (Estado)
String dni;
String nombre;
String apellidos;

// CONSTRUCTOR
// Se ejecuta al hacer: new Empleado("12345678X", "Brais", "García")
public Empleado(String dni, String nombre, String apellidos) {
    this.dni = dni;           // 'this.dni' es el atributo, 'dni' es el parámetro
    this.nombre = nombre;
    this.apellidos = apellidos;
}

// Método para mostrar info
public void presentarse() {
    System.out.println("Empleado: " + nombre + " " + apellidos + " (DNI: " + dni + ")");
}
}
```

Ejemplo de uso

```
public class GestionEmpresa {
  public static void main(String[] args) {
      // Aquí usamos 'new' para reservar memoria
      // Y llamamos al 'constructor' pasándole los valores
      Empleado emp1 = new Empleado("12345678X", "Brais", "García");

      emp1.presentarse();
  }
}
```

---

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

this es una palabra reservada que representa una referencia a la instancia actual del objeto que está ejecutando el código.

Sus usos principales son:

Desambiguación: Diferenciar entre un atributo de la clase y un parámetro del método si ambos se llaman igual (sombreado de variables).

Pasar el objeto actual: Poder pasar "el objeto entero" como argumento a otro método.

Llamar a otros constructores: Dentro de un constructor, usar this() para invocar a otro constructor de la misma clase.

No se usa igual en todos los lenguajes, aunque el concepto de "auto-referencia" existe en casi todos los lenguajes de POO, el nombre cambia:

this: Es el estándar en Java, C++, C#, JavaScript y PHP.

self: Es el término utilizado en Python (donde además es obligatorio pasarlo como primer parámetro de los métodos) y Swift.

Me: Es el término utilizado en Visual Basic .NET.

- **Ejemplo:**

  public class Punto {
  double x;
  double y;

```
// Uso de 'this' para evitar la ambigüedad
public Punto(double x, double y) {
    this.x = x; // El valor del parámetro 'x' se asigna al atributo 'x' de esta instan
    this.y = y; // 'this.y' es el atributo del objeto, 'y' es el dato que entra por el
}

// Otro uso: un método que mueve el punto a otra ubicación
public void moverA(double x, double y) {
    this.x = x;
    this.y = y;
}

public double calculaDistanciaAOrigen() {
```

```
    // Aquí 'this' es opcional pero ayuda a la legibilidad
    return Math.sqrt(Math.pow(this.x, 2) + Math.pow(this.y, 2));
}
}
```

---

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

public class Punto {

double x;

double y;

```
// Constructor que ya conocemos
public Punto(double x, double y) {
    this.x = x;
    this.y = y;
}

// Método original (distancia al 0,0)
public double calculaDistanciaAOrigen() {
    return Math.sqrt(Math.pow(this.x, 2) + Math.pow(this.y, 2));
}

// NUEVO MÉTODO: Distancia entre este punto (this) y otro punto recibido
public double distanciaA(Punto otroPunto) {
    // (x2 - x1) -> otroPunto.x - this.x
    double diferenciaX = otroPunto.x - this.x;
    // (y2 - y1) -> otroPunto.y - this.y
    double diferenciaY = otroPunto.y - this.y;

    return Math.sqrt(Math.pow(diferenciaX, 2) + Math.pow(diferenciaY, 2));
}
}
```

---

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?

Cuando pasas un objeto Punto a un método, lo que estás copiando y pasando es la referencia (la dirección de memoria).

¿Afectan los cambios? SÍ. Como tanto la variable de fuera del método como la de dentro apuntan al mismo objeto en el Heap, si cambias un atributo (como punto.x = 10), el cambio persiste fuera.

La excepción: Si dentro del método haces punto = new Punto(), ahí estás cambiando la dirección de la copia local. A partir de ese momento, los cambios que hagas ya no afectarán al punto de fuera, porque la variable local ahora apunta a un objeto nuevo.

Con los tipos primitivos (int, double, boolean, char), Java copia el valor real del dato (el número 5, por ejemplo) en una nueva posición del Stack.

¿Afectan los cambios? NO. El método recibe una copia exacta del número. Si dentro del método haces entero = 100, solo estás modificando esa copia local. La variable original fuera del método seguirá valiendo lo que valía antes.

---

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

toString() es un método definido originalmente en la clase Object (la clase "madre" de todas las clases). Su propósito es devolver una representación en forma de cadena de texto (String) del contenido de un objeto.

Comportamiento por defecto: Si no lo escribes tú, Java usa el de la clase Object, que devuelve algo feo como: NombreDeClase@DireccionDeMemoria (ej. Punto@1b6d3586).

Sobrescritura (@Override): Lo normal es que nosotros lo "sobrescribamos" para que devuelva los datos reales del objeto (sus atributos).

- **¿Existe en otros lenguajes?**

  Sí, casi todos los lenguajes modernos tienen un equivalente, porque imprimir objetos para depurar (debugging) o mostrar información es una necesidad universal.

- **Ejemplo:**

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

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

  // SOBRESCRITURA DEL MÉTODO toString
  @Override
  public String toString() {
      // Devolvemos una cadena con el formato que más nos guste
      return "Punto[x=" + this.x + ", y=" + this.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?

- **¿Es una clase como un struct?**

Sí, pero solo a medias. Ambos son tipos de datos definidos por el usuario que sirven para agrupar variables de distintos tipos bajo un mismo nombre. Podríamos decir que una clase es un struct "con esteroides".

- **¿Qué le falta al struct para ser una clase?**

Para que un struct se convierta en una clase y sus variables sean verdaderas instancias, le faltan principalmente tres cosas:

Comportamiento (Métodos): Un struct solo guarda datos (estado). Una clase guarda datos y también las funciones (métodos) que operan sobre esos datos. En C, los datos y las funciones están separados; en Java, están unidos en el objeto.

Encapsulamiento (Privacidad): En un struct, todos los campos son públicos y cualquiera puede modificarlos. La clase permite usar private para proteger sus entrañas, obligando a usar métodos controlados.

Herencia y Polimorfismo: Un struct no puede heredar de otro ni responder de formas distintas a una misma llamada. No permite crear jerarquías de tipos.

- **¿Qué le falta para que sus variables sean "instancias"?**

En C, cuando declaras un struct, manejas la memoria de forma muy manual (estática o mediante punteros). Para que se comporte como una instancia de Java:

Gestión de Memoria Automática: En Java, la "instancia" vive en el Heap y el Garbage Collector la limpia. En C, tú eres el responsable de liberar la memoria (o se libera sola si es local).

Abstracción de la Identidad: En Java, tratamos a la instancia a través de una referencia transparente. En C, tienes que lidiar con la dirección de memoria física (punteros) de forma explícita (* y &).

---

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?

Para emular una clase de Java en C, tenemos que hacer manualmente lo que Java hace de forma automática. En C, las funciones no pueden vivir dentro de los datos, así que tenemos que "conectarlas" nosotros.

- **1. El Estado (El struct)**

Primero definimos el contenedor de datos, igual que en Java, pero aquí solo hay variables.

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

- **2. El Comportamiento (La función externa)**

Como la función no puede estar dentro del struct, la creamos fuera. Pero, ¿cómo sabe la función qué punto específico debe usar? Aquí es donde aparece nuestro this manual.

Para emular a this, pasamos un puntero al struct como primer argumento de la función. Por convención, en C se le suele llamar self o this.

```
#include <math.h>

double calculaDistanciaAOrigen(Punto* self) {
    // Usamos 'self->' para acceder a los datos del objeto apuntado
return sqrt(pow(self->x, 2) + pow(self->y, 2));
```

}

- **3. El uso (La emulación de la "instancia")**

En el main, tenemos que pasar explícitamente la dirección de memoria del "objeto" a la función.

```
int main() {
    Punto miPunto; // "Instancia"
    miPunto.x = 3.0;
    miPunto.y = 4.0;

    // Llamada manual: pasamos la dirección (&) para que sea el 'self'
    double d = calculaDistanciaAOrigen(&miPunto);

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

```
    return 0;
}
```

- **Reflexión: ¿Qué ha pasado con this?**

Lo que acabamos de hacer manualmente es exactamente lo que hace Java por ti de forma invisible:

En Java, cuando escribes miPunto.calculaDistanciaAOrigen(), el compilador de Java en realidad transforma esa llamada en algo muy parecido a lo que hemos hecho en C.

El "This" oculto: Java pasa la referencia del objeto (miPunto) como un parámetro invisible (el parámetro 0) a todos los métodos que no sean static.

La palabra reservada this es simplemente el nombre que Java le da a ese parámetro invisible para que tú puedas usarlo en tu código si lo necesitas.
