MET. DE LA PROGRAMACIÓN

Pablo Sevilla Méndez

—LENGUAJES DE PROGRAMACIÓN—

2 tipos de lenguajes:

- Naturales (Español,inglés)
- Formales (Lenguajes de programación...). Un lenguaje formal es un lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos están formalmente especificados.

Lenguajes formales:

• **Lenguajes Máquina** – Son los lenguajes de más bajo nivel: secuencias binarias de ceros y unos. – Históricamente, los primeros	• **Lenguajes Ensambladores** – Segunda generación de lenguajes – Versión simbólica de los lenguajes máquina (MOV, ADD, etc).
• **Lenguajes de Alto Nivel** – Lenguajes de tercera generación (3GL) - Estructuras de control, Variables de tipo, Recursividad, etc. - Ej.: C, Pascal, C++, Java, etc	•**Lenguajes Orientados a Problemas**: Describen la solución, no cómo conseguirla – Lenguajes de cuarta generación (4GL) Ej. SQL

Traductores→ intérpretes y compiladores:

Traductor: Un traductor es un programa que lee un programa escrito en lenguaje **fuente** de alto nivel, y lo traduce a un lenguaje **objeto**.

Según sus funcionalidades puede ser de dos tipos:

– **Intérprete**: Ejecuta directamente lo que traduce, sin almacenar en disco la traducción realizada.

1. Cada vez que se escribe una línea el programa comprueba si es correcta, si lo es, la ejecuta.

2. La ejecución es interactiva.

3. Los intérpretes más puros no guardan copia del programa que se está escribiendo

– Ejemplos: Procesos por lotes, JavaScript, CAML, etc.

– El intérprete está siempre presente para la traducción y ejecución

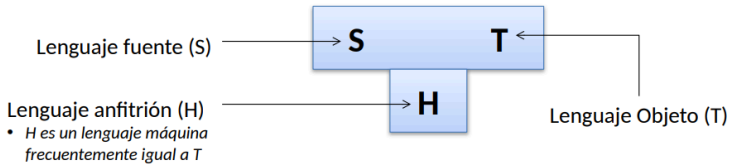
– **Compilador**: Genera un fichero del lenguaje objeto estipulado, que puede posteriormente ser ejecutado tantas veces se necesite sin volver a traducir.

• Como beneficio adicional, el compilador informa de los errores del programa fuente, ya que lo analiza al completo.

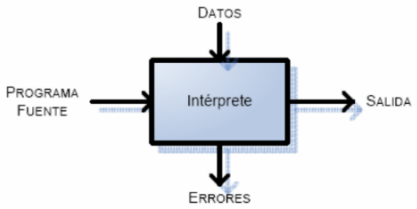
Un compilador es un programa que lee un programa escrito en lenguaje fuente, y lo traduce a un lenguaje objeto de bajo nivel. Además, genera una lista de los posibles errores que tenga el programa fuente.

– El compilador es el traductor más extendido

– El programa ejecutable, una vez creado, no necesita el compilador para funcionar



• *H es un lenguaje máquina frecuentemente igual a T*



—POO Teoría—

Desarrollamos los siguientes conceptos:

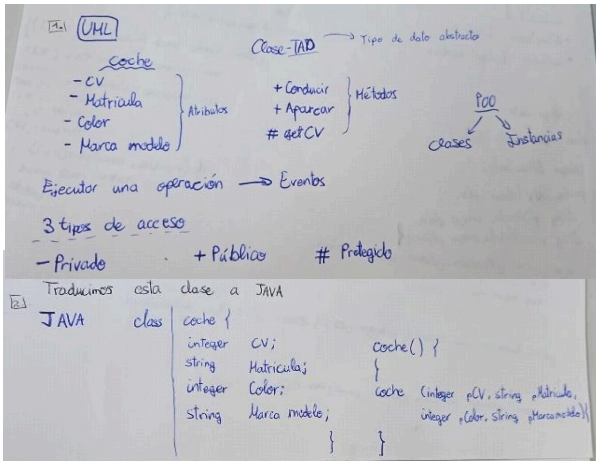
-<u>Abstracción</u>: se refiere a la capacidad de representar y manejar conceptos complejos de manera simplificada. La abstracción en programación está estrechamente relacionada con el paradigma de programación orientada a objetos (POO). En la POO, los conceptos del mundo real se modelan como objetos, y la abstracción consiste en definir interfaces y comportamientos comunes para estos objetos. Para implementar la abstracción en la POO hacemos uso de clases abstractas. Una **clase abstracta** es una plantilla que define un conjunto de métodos y propiedades comunes para un grupo de objetos relacionados. En cambio, las **clases concretas**, que heredan de la clase abstracta, proporcionan implementaciones específicas de estos métodos y propiedades.

-La **<u>programación modular</u>** se refiere a un estilo de desarrollo de software en el que un programa se divide en módulos o componentes más pequeños y manejables. Cada módulo realiza una tarea específica y se comunica con otros módulos a través de interfaces definidas. Estos módulos pueden ser funciones, clases, …etc , dependiendo del lenguaje de programación utilizado.

- **<u>Herencia</u>**: mecanismo por el cual una clase hereda los atributos y métodos de otra clase.

- **<u>Polimorfismo</u>**: es una técnica en la programación orientada a objetos que permite que los objetos de diferentes clases respondan a un mismo mensaje de diferentes maneras. Esto significa que el mismo método puede tener diferentes comportamientos según la clase del objeto que lo recibe.

- <u>UML</u>: lenguaje modelado unificado, es un lenguaje gráfico (no es de programación) y su función es describir el proceso completo de ingeniería del software desde la captura de requisitos hasta que tenemos el programa instalado en una máquina y en ejecución.



Estoy sobrecargando el nombre de una función, no estoy escribiendo dos veces la misma función, la condición es que no puedo hacerlo en el mismo orden. También puedo hacer

sobrecarga de operaciones. El primero es un constructor por defecto (no tiene ningún parámetro), su función es inicializar los valores y cualquier otra operación que se pueda necesitar.

Destructores: método que se va a llamar cuando una instancia de esa clase se elimina de memoria, trabajan de distinta manera dependiendo del lenguaje en java no se utilizan casi nunca, en java no hace falta liberar memoria.

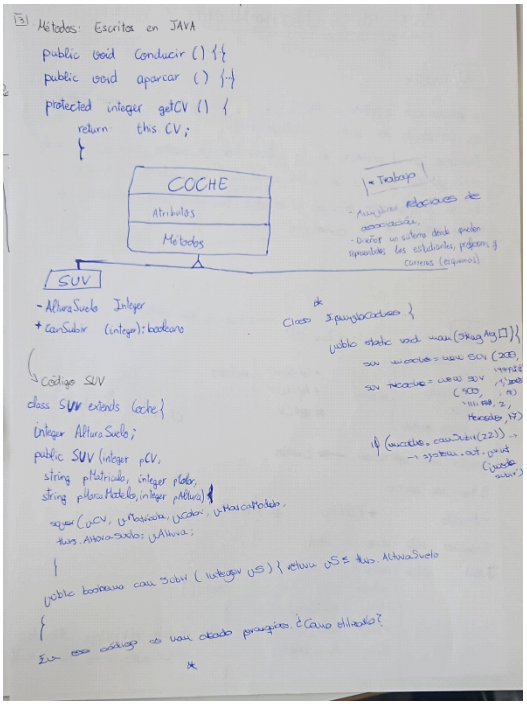
Metodos: escritos en Java;

public void conducir() {...} *void significa sin devolucion

public void aparcar () {...}

protected integer getCV() {...}

return this.cv



*Class SUV **extends** Coche {*

integer AlturaSuelo

public suv (integer pCV, String pMatricula, integer pColor, String pMarcaModelo, i integer pAltura)}

Extends: se utiliza en la programación orientada a objetos para establecer una relación de herencia entre clases.

Integer: clase envolvente que proporciona un conjunto de métodos estáticos y de instancia para trabajar con valores enteros de tipo primitivo int.

Super:

Referencia a un miembro de la clase base: Cuando una clase hereda de otra clase, se puede utilizar "super" para acceder a los métodos de la clase base desde la clase derivada. Además, en el constructor de una clase derivada, "super" se utiliza para invocar explícitamente el constructor de la clase base. Esto se hace para inicializar las partes de la clase base de manera adecuada antes de la inicialización de las partes específicas de la clase derivada.

Para que sirve hacer esto en java:

SUV micoche =new SUV(2005,34567,1,"hdgchsd", 18)

Creamos una instancia de un objeto SUV con ciertas características:

'suv': Creamos un objeto de la clase Suv, esta es una clase definida en nuestro código.

'micoche': es el nombre de la variable que hace referencia al objeto SUV recién creado.

'new SUV(2005,34567,1,"hdgchsd", 18): new se utiliza para reservar memoria para un nuevo objeto SUV en el montón de memoria y llama al constructor de la clase SUV. Entre paréntesis se pasan los argumentos al constructor. En este caso, parece que el constructor de SUV toma cinco argumentos.

Función del **void**: En el lenguaje de programación Java, el uso de la palabra reservada void es crucial para definir un método que no retorna ningún valor

Anotación: los elementos que llevan un @ delante y están fuera de los métodos y clases

Override: lo que indicamos con esto es que el compilador Java debe asegurarse de que existe un método con la misma firma que el que estamos definiendo en la superclase de la clase que estamos definiendo. Esto ayuda a no equivocarnos al escribir los nombres y firmas de los métodos a redefinir .Al usar @Override, lo que estamos haciendo es asegurarnos que hemos redefinido bien el método. Hay que marcar como override los métodos que se sobrecarguen.

El **método toString** nos permite mostrar la información completa de un objeto, es decir, el valor de sus atributos.

La diferencia entre **float y Float** es: El primero es un tipo básico y el segundo un nombre de clase.

— POO-UML Práctica—

(GitHub)

1. GITHUB:

Git es un Sistema de Control de Versiones (VCS) distribuido, lo que significa que es una herramienta útil para identificar fácilmente los cambios en tu código, colaborar y compartir. Con Git puedes hacer un seguimiento de los cambios que haces en tu proyecto, de modo que siempre tienes un registro de lo que has trabajado y puedes volver fácilmente a una versión anterior si es necesario. También facilita el trabajo en equipo: grupos de personas pueden trabajar juntas en el mismo proyecto y fusionar sus cambios en un único código fuente final.

1.1.Funcionamiento de GitHub:

-**Repositorios**: Contiene todos los archivos de tu proyecto y el historial de revisiones. Piensa en un repositorio como la carpeta de su proyecto. Puedes invitar a otros a colaborar contigo en estos archivos.

-**Cloning**: Cuando se crea un repositorio con GitHub, se almacena remotamente en la nube. Puedes clonar un repositorio para crear una copia local en tu ordenador y luego utilizar Git para sincronizar ambos. Esto facilita la corrección de problemas, la adición o eliminación de archivos y el envío de commits más grandes.

-**Commiting and pushing**: son la manera de añadir los cambios que has hecho en tu máquina local al repositorio remoto en GitHub. De esta manera tu instructor y/o compañeros de equipo pueden ver tu último trabajo cuando estés listo para compartirlo.

-Branches: puedes utilizar ramas en GitHub para aislar el trabajo que no quieres fusionar en tu proyecto final todavía. Las ramas te permiten desarrollar características, corregir errores o experimentar con nuevas ideas de forma segura en un área contenida de tu repositorio.

-**Pull request**: Informa a los demás de los cambios que quieres hacer y les pides su opinión acerca de esos cambios.

-**Issues**: son una forma de hacer un seguimiento de las mejoras, tareas o errores de tu trabajo en GitHub.

(Conceptos)

Conceptos básicos en POO:

CONSTRUCTOR:

Se encarga de generar un objeto de una clase con el objetivo de inicializarlo para su uso posterior, dándole valores a sus atributos y modificando configuraciones necesarias. Puede recibir parámetros para modificar los atributos o no recibirlos.

GETTER:

Se trata de un método encargado de retornar el valor de un atributo. Los atributos generalmente son privados por lo que usaremos los getters para devolver los valores de los atributos que queramos, ya que desde el exterior no podrán consultarlos.

PUBLIC VS PRIVATE VS PROTECTED:

Para determinar quiénes pueden acceder a los datos de una variable, función… etc. tenemos principalmente dos opciones. “Public” hará que dicho elemento pueda ser accesible desde cualquier parte del programa e incluso se podrá consultar su información desde el exterior. “Private” hará que dicha variable o función pueda ser llamada únicamente desde su misma clase, ni siquiera otras clases de ese mismo paquete podrán utilizar dicho elemento. Por último, contamos con “Protected”, que al aplicarlo a un elemento hará que este pueda ser llamado desde su clase, clases del mismo paquete e incluso subclases de otros paquetes. Contribuye a la seguridad (está bloqueado el acceso a algunos usuarios) y al diseño modular del código.

SETTER:

Los setter actúan como puerta para recibir datos nuevos y asignarlos a un atributo. Antes de ello también se asegura que son del tipo necesario para dicho atributo y de si el valor que queremos establecer es válido para las operaciones asignadas a ese atributo.

KEYWORD “THIS”:

Las ‘keywords’ son palabras reservadas en un lenguaje para realizar una función especial. “this” se encarga de definir sobre qué variables trabaja una función, lo cual es útil cuando una función puede actuar con varios objetos de la misma clase. Es recomendable usarlo para concretar a qué ámbito de variable se refiere el programador.

MÉTODO toString() :

En java podemos asignar un método público para convertir los objetos de una clase en una cadena de texto con su descripción. Se puede llamar de forma explícita o de forma implícita.

—POO-UML Teoría—

ASOCIACIONES

Las asociaciones nos permiten que clases distintas se relacionen entre ellas.

→CARACTERÍSTICAS PRINCIPALES:

- Facilita la lectura de relaciones
- **Multiplicidad**→ Cuántas veces puede aparecer esa asociación. Actúa en ambas direcciones. Tiene como posibles valores 0, 1, n (excepcionalmente un número) Un 0 significa que esta asociación puede no existir en alguna dirección.

Hemos visto que una relación entre dos clases nos permite establecer un **atributo**[roll], y eso nos permite establecer un primer diseño de cómo funcionan las cosas. El atributo puede venir de cualquiera de los dos sitios si cambia la multiplicidad.
.
Existe la posibilidad de tener una punta de flecha en cada sentido, pero no se usa. Lo que se usa es una segunda flecha con su multiplicidad y con un rol alternativo. Porque que un elemento enganche a otro en un lado, no asegura estructuralmente que ocurra recíprocamente, por eso establecemos dos asociaciones distintas.

DIAGRAMA DE SECUENCIAS→ Donde se plasma la ejecución de algo, se plasma en función de la secuencia de operaciones que se realizan

Diferencias entre DIAGRAMA DE CLASES Y DE SECUENCIAS:
El diagrama de clases en un diagrama estático, te da la estructura con los objetos de tu aplicación en funcionamiento. Mientras que el diagrama de secuencias te da la parte dinámica, puesto que es donde se plasma la ejecución de las operaciones que se realizan en función de su secuencia. Expresa cómo se comporta el diagrama de clases

—INTERFACES Y PLANTILLAS—

INTERFACES

Cuando creamos una nueva clase a partir de una anterior, y esta tiene ciertas cualidades y atributos, la herencia nos permite que esta nueva clase herede los métodos y atributos de su clase (o clases, en caso de herencia múltiple) predecesora. Las interfaces se implementaron para que no surja un conflicto si dos clases que proceden de distintas herencias se llaman igual. Las interfaces están construidas con las firmas de los métodos públicos que queremos que existan, solo las firmas y no el código. Además nos permite elegir en caso de herencia múltiple. Al implementar una interfaz solo declararemos las clases de los métodos. Luego sólo con la interfaz sin importar la clase instanciamos sus parámetros, para darle valores más adelante.

public interface Pez { *public String tipo();* *public String toString();* *}*	*public interface Aterrestre {* *public int getNombrePatas();* *}*

Si las interfaces tienen la misma definición de un método, no pasa nada. Se supone que tenemos que haber definido correctamente los parámetros de nuestra interfaz. Dentro de la nueva clase que contendrá las interfaces podemos definir nuevos métodos.

```
public class Anfibio implements Pez, Aterrestre{
        public String tipo(){    }
        public String toString(){   }
        public int getNumPatas(){   }
}
```

PARAMETRIZACIÓN

En java, el uso de parámetros en métodos, clases o interfaces se conoce como parametrización. Se puede trabajar con genéricos, lo cual mejora la flexibilidad y la seguridad del código. Su objetivo es que el código sea utilizado con diferentes tipos de datos sin tener que reescribirlo.

—HERENCIA—

La herencia se refiere a la práctica, en el contexto de la programación, de crear "**subclases**" sobre clases previamente creadas, a las que llamaremos "**superclase".** Las subclases heredan de las superclases sus atributos y métodos. La subclase invoca al constructor de la superclase usando el método super().

Sobrecarga de métodos y "super": La sobrecarga de métodos permite que un mismo método sea definido en dos clases relacionadas por herencia. La superclase tendrá una definición e implementación del método, mientras que la subclase tendrá otra.

El problema en este caso es cómo hacer que la subclase pueda invocar al método de la superclase. Para ello tenemos la palabra reservada "**super**", que se refiere a la parte de la instancia definida por la superclase.

Si queremos usar un método establecido tanto en la super como en la subclase, es decir, sobrecargado, deberemos usar: **super.#####()** Ya que de lo contrario, java no sabrá si acceder a una u otra, y esto puede derivar en problemas.

Si se marcan los atributos como privados, no se puede acceder a ellos desde la subclase:

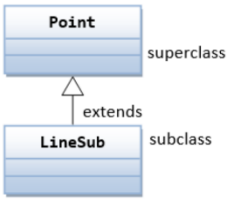
Hay que usar los getters y setters correspondientes, por ejemplo:

```
public void setXYZ(float x, float y, float z) {
    setX(x);     // or super.setX(x), use setter in superclass
    setY(y);
    this.z = z;
}
```

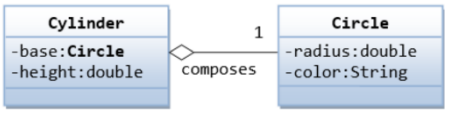
—COMPOSICIÓN—

Esta se refiere a la práctica de usar clases previamente creadas, para dar lugar a objetos con los que posteriormente, en el constructor, hacer uso de estos. Creamos así un objeto compuesto de otros, que pasan a ser sus atributos. Esto puede permitir que se varíe la estructura en tiempo de ejecución mediante el uso de listas de objetos que pueden crecer y decrecer. En el caso que nos ocupa, eso no sucede, ya que se especifica que "**una línea se compone de dos puntos"**: *sin esos puntos, no se puede componer*. Cosa distinta es la agregación, que sí permite la variación de esos elementos. (NOTA: composición = rombo relleno, agregación = rombo en blanco).

Si ahora intentamos el diseño mediante la herencia, una clase Line heredará, de manera que la definición del problema quedará como: "*Una línea es un punto extendido por otro punto*", o dicho de otro modo: una línea tiene un punto de origen, e internamente tiene un punto de fin:



En conclusión; la composición es simplemente usar en una clase, otra clase previamente creada para construir el objeto de nuestra nueva clase; como en este ejemplo:




```
public class Cylinder {
    private Circle base;   // Base circle, an instance of Circle class
    private double height;

    // Constructor with default color, radius and height
    public Cylinder() {
        base = new Circle(); // Call the constructor to construct the Circle
        height = 1.0;
    }
    ......
```

CLASES ABSTRACTAS

Una clase abstracta Java, es una clase especial que no se puede instanciar. Dicha clase está destinada a ser usada como base para otras clases, que se conocen como subclases.

Se usa para definir atributos y métodos comunes que pueden ser compartidos por todas las clases que heredan de la clase abstracta. Los métodos definidos en la clase abstracta generalmente deben ser implementados en las subclases.

Una clase abstracta puede contener métodos abstractos, que son declaraciones sin implementación. De hecho, estos métodos deben ser implementados por las subclases para ser usados. Esto permite a los desarrolladores definir una funcionalidad general y luego dejar que cada subclase implemente la funcionalidad de manera diferente.

Las clases abstractas también se pueden usar para definir una interfaz para un conjunto de clases. Esto significa que todas las clases que implementan la interfaz tendrán los mismos métodos, aunque el comportamiento de los métodos pueda ser diferente. Es una característica muy útil cuando se quiere garantizar que un conjunto de clases tenga un comportamiento similar.

INTERFACES

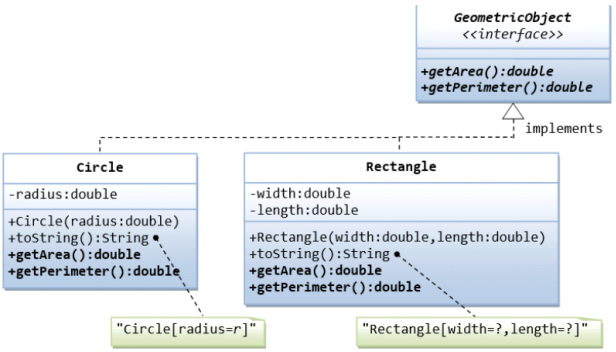
Un interfaz representa un contrato público. Aquellas clases que quieran disfrutar de un interfaz, deben implementar todos losmétodos indicados. Los programas que usen un interfaz permitirán que cualquier clase de objetos que implemente el interfazpueda ser manejado de manera transparente.

Todas las clases que lleven implementado un interfaz deben llevar los metodos que dicha interfaz tenga. (ej: geometricobject(){getperimeter, getarea} entonces public circle implements geometric object"debe de contener getarea y getperimeter como metodos dentro"(){getarea, getperimeter}

*denotates package access--->indica que los miembros son accesibles solo dentro del mismo paquete

FUNCION DE IMPLEMENTS, EXTENDS

1. Implements



En Java, la palabra reservada “implements” se utiliza para implementar una interfaz en una clase. Una interfaz define un conjunto de métodos que una clase debe implementar. Al utilizar “implements”, una clase adquiere todos los métodos definidos en la interfaz, lo que le permite cumplir con un contrato específico.

public class MiClase implements MiInterfaz {

// Implementación de los métodos definidos en la interfaz

}

2.Extends

Se utiliza para establecer una relación de herencia entre clases. Permite que una clase herede los campos y métodos de otra clase, conocida como clase padre o superclase, lo que facilita la reutilización de código y la implementación de jerarquías de clases.