

METODOLOGÍA DE LA PROGRAMACIÓN ~CUADERNO DE TRABAJO~

UNIVERSIDAD DE ÁLCALÁ- GRADO EN MATEMÁTICAS Y COMPUTACIÓN

CURSO: 2023/2024

ALUMNOS:

GARRO OLMEDA, DIEGO

ZOUFRI GARCIA, SAMIRA

ÍNDICE

	N.º de página
Día 1	3
Día 2	8
Día 3	11
Día 4	14
Día 5	17
Día 6	19

El cuaderno quedará dividido por días de clase. En cada día constará, por una parte, el código y explicación en clase y por otra, una explicación de cosas que no entendíamos.

1ª Clase (24/01/2024)

¿Qué es un COMMIT?

Acción de guardar de forma permanente los cambios realizados en un proyecto. Nosotros utilizaremos GitHub, que además nos permite seguir el historial de modificaciones en códigos (de esta manera hay mejor registro para ver quién y cuando hizo algún cambio)

¿Qué es un VCS?

(Sistema de control de versiones). Es un software que os permite almacenar y gestionar diferentes versiones de archivos. Cada versión se guarda como un "COMMIT", y guarda bastante información, como el autor, la fecha y hora de la modificación. Además, nos permite llevar un seguimiento de los cambios realizados en los archivos, volver a una versión anterior si algo sale mal incluso colaborar con otros usuarios. Un ejemplo es el ya mencionado GitHub.

¿Qué es un VSS?

(Visual SourceSafe). Es un sistema de control de versiones específico, aunque ahora ha sido remplazado por Git. Tiene diferentes desventajas como falta de flexibilidad, incompatibilidad con el desarrollo distribuido y menor seguridad.

¿Qué es RSYNC?

Es una herramienta utilizada para la transferencia y sincronización eficiente de archivos y directorios entre sistemas locales y remotos. Sus características principales incluyen;

- Sincronización eficiente: Rsync que compara archivos por tamaño y fecha de modificación en lugar de todo archivo.
- Versatilidad: funciona en sistemas locales y remotos, incluso a través de conexiones comprimidas o cifradas.
- Ofrece numerosas opciones para definir criterios de sincronización, como preservación de permisos y propiedades.
- Seguridad: puede emplear autenticación y cifrado para proteger la transmisión de datos
- Es gratuito y multiplataforma.

¿Fetch?

Es una API (interfaz de Programación de Aplicaciones) moderna que permite obtener recursos de la red de forma asincrónica, como datos de servidores o archivos externos. Tiene una sintaxis simplificada y flexible, para facilitar la comunicación entre diferentes dominios, y una amplia compatibilidad con navegadores principales

¿Qué es merge?

En git implica combinar dos ramas o versiones diferentes de un proyecto en una sola línea de desarrollo. Es útil cuando múltiples colaboradores han estado trabajando en cambios independientes y se desea unirlos en una única rama

¿Windows, Mac y Linux?

Un sistema operativo es un programa o conjunto de programas que gestiona los recursos de hardware y software de un dispositivo informático. Es el software fundamental que permite el funcionamiento del ordenador y la ejecución de otros programas.

Hay diferentes tipos: Window, MacOS, Linux, Android etc.

Pero a nosotros nos interesan 3 tipos:

Windows:

Pros:

- Amplia compatibilidad: Es el sistema operativo más popular y es compatible con la mayoría del software y hardware disponible.
- Facilidad de uso: Cuenta con una interfaz gráfica intuitiva y familiar para muchos usuarios, lo que facilita su utilización.
- Gran cantidad de software disponible: Ofrece una amplia gama de aplicaciones para satisfacer diferentes necesidades de los usuarios.
- Soporte técnico: Dispone de una amplia red de soporte técnico oficial y de terceros para resolver problemas y brindar ayuda.

Contras:

- Costo: Las licencias de Windows suelen ser más costosas que las de otros sistemas operativos, lo que puede representar un gasto adicional para los usuarios.
- Menos personalizable: Ofrece menos flexibilidad en comparación con otros sistemas operativos para personalizar la apariencia y el funcionamiento del sistema según las preferencias individuales.
- Seguridad: Puede ser más vulnerable a ataques de malware y virus, lo que puede comprometer la seguridad y privacidad de los usuarios.
- Rendimiento: En algunos casos, puede ser menos eficiente que otros sistemas operativos en términos de rendimiento y uso de recursos del sistema.

MAC

Pros:

- Diseño elegante: Tanto el hardware como el software de Mac tienen un diseño atractivo y cuidado.
- Facilidad de uso: macOS cuenta con una interfaz gráfica intuitiva y conocida por su simplicidad.
- Buena integración con otros productos Apple: Los dispositivos Mac funcionan perfectamente con iPhones, iPads y otros productos de Apple.
- Seguridad: macOS tiene una reputación de ser un sistema operativo seguro.

Contras:

- Costo: Los productos de Apple tienden a ser más costosos que los de la competencia.
- Menos software disponible: Hay una menor cantidad de software disponible en comparación con Windows.
- Menos personalizable: Ofrece menos flexibilidad para personalizar el sistema operativo.
- Soporte técnico: La red de soporte técnico de Mac es más limitada que la de Windows.

Linux:

Pros:

- Gratuito y de código abierto: Linux es un sistema operativo libre y de código abierto, lo que significa que es gratuito y puede ser modificado y distribuido libremente.
- Personalizable: Linux es altamente personalizable, permitiendo adaptar el sistema operativo a las necesidades específicas del usuario.
- Seguridad: Linux es considerado uno de los sistemas operativos más seguros.
- Eficiencia: Linux generalmente tiene un rendimiento más eficiente que Windows y macOS en algunos casos.

Contras:

- Curva de aprendizaje: Puede ser menos intuitivo para usuarios principiantes.

- Compatibilidad: Linux tiene menor compatibilidad con software y hardware en comparación con Windows.
- Soporte técnico: La red de soporte técnico de Linux es menos extensa que la de Windows o macOS.
- Menos software disponible: Hay una menor cantidad de software disponible en comparación con Windows.

Teoría

Modificadores de acceso en Java: pública, privada y protegida

Introducción:

Los modificadores de acceso en Java controlan quién puede acceder a los miembros de una clase (atributos y métodos).

Tipos de modificadores:

Public: Permite el acceso desde cualquier clase, incluso en diferentes paquetes. Se usa para miembros que deben ser accesibles desde cualquier parte del programa.

Private: Restringe el acceso al miembro a la clase en la que se define. Se usa para miembros que solo necesitan ser utilizados por la clase en sí.

Protected: Permite el acceso desde la clase en la que se define y desde sus subclases. Se usa para miembros que deben ser accesibles a las subclases, pero no a otras clases.

Default (paquete): Otorga acceso a los miembros dentro del mismo paquete. Se usa para miembros que no necesitan ser públicos, pero sí accesibles a otras clases del mismo paquete.

Resumen:

Modificador	Visibilidad
Public	Accesible desde cualquier lugar.
Private	Solo accesible desde la clase que lo define.
Protected	Accesible desde la clase que lo define y sus subclases.
Default	Accesible dentro del mismo paquete.

Recomendaciones:

- Se recomienda usar el modificador más restrictivo que cumpla con los requisitos de acceso.
- El uso adecuado de los modificadores de acceso mejora la seguridad y el encapsulamiento del código.

Ejemplo:

```
public class Persona {  
    public String nombre; // Public - Accesible desde cualquier lugar  
    private int edad; // Private - Solo accesible desde la clase Persona  
    protected void saludar() { // Protected - Accesible desde Persona y sus subclases  
        System.out.println("Hola, mi nombre es " + nombre);  
    }  
    // Default - Accesible dentro del mismo paquete  
    void mostrarEdad() {
```

```
System.out.println("Mi edad es " + edad);  
}  
}
```

En este ejemplo:

- nombre** es un atributo público, por lo que se puede acceder a él desde cualquier clase.
- edad** es un atributo privado, por lo que solo se puede acceder a él desde la clase `Persona`.
- saludar** es un método protegido, por lo que solo se puede acceder a él desde la clase `Persona` y sus subclases.
- mostrarEdad** es un método con el modificador de acceso por defecto, por lo que solo se puede acceder a él desde las clases del mismo paquete.

Conclusión:

Los modificadores de acceso son una herramienta importante para controlar la accesibilidad de los miembros de una clase. Es importante comprender cómo funcionan y elegir el modificador adecuado para cada caso.

¿Qué es una máquina virtual?

Una máquina virtual (VM) es un software que simula un equipo físico completo. Se ejecuta dentro de un sistema operativo anfitrión y proporciona un entorno aislado para ejecutar otro sistema operativo, aplicaciones y archivos.

En otras palabras:

- Es como un ordenador dentro de otro ordenador.
- Tiene su propio CPU, memoria, almacenamiento y red.
- Puede ejecutar cualquier sistema operativo que sea compatible con el software de virtualización.

Para qué se utilizan las máquinas virtuales:

- Probar software:** Puedes probar software en una VM sin afectar a tu sistema operativo principal.
- Ejecutar diferentes sistemas operativos:** Puedes ejecutar Windows, macOS y Linux en el mismo ordenador.
- Aislar entornos:** Puedes crear una VM para cada proyecto o tarea para mantenerlos separados.
- Ahorrar recursos:** Puedes ejecutar varios sistemas operativos en un solo ordenador físico.
- Desarrollar software:** Puedes crear un entorno de desarrollo aislado para cada proyecto.

Tipos de máquinas virtuales:

- Máquinas virtuales de escritorio:** Se utilizan para ejecutar aplicaciones en un ordenador personal.
- Máquinas virtuales de servidor:** Se utilizan para ejecutar servidores en un centro de datos.
- Máquinas virtuales en la nube:** Se utilizan para ejecutar aplicaciones en la nube.

Software de virtualización:

- VMware:** Es uno del software de virtualización más populares.
- VirtualBox:** Es un software de virtualización gratuito y de código abierto.
- Hyper-V:** Es un software de virtualización gratuito de Microsoft.

Ventajas de las máquinas virtuales:

Flexibilidad: Puedes ejecutar diferentes sistemas operativos y aplicaciones en el mismo ordenador.

Aislamiento: Puedes aislar entornos para mayor seguridad y estabilidad.

Portabilidad: Puedes mover las VMs fácilmente entre diferentes ordenadores.

Eficiencia: Puedes ahorrar recursos ejecutando varios sistemas operativos en un solo ordenador.

Desventajas de las máquinas virtuales:

Complejidad: Configurar y administrar las VMs puede ser complejo.

Rendimiento: Las VMs pueden tener un menor rendimiento que los equipos físicos.

Costo: El software de virtualización puede ser costoso.

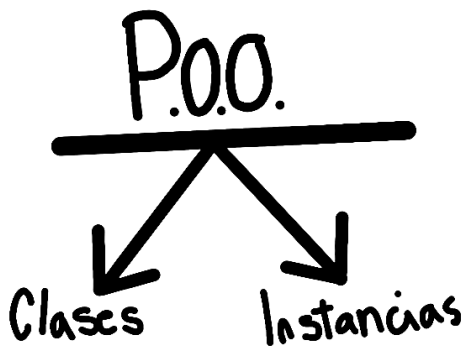
¿Qué es un diagrama UML?

Un diagrama UML, o Diagrama de Modelado Unificado, es una representación visual de los conceptos y estructuras de un sistema. Además, representan diferentes aspectos de un sistema de software, incluyendo su estructura estática, comportamiento dinámico, interacciones entre componentes, casos de uso, clases, objetos, relaciones, estados, actividades, y más. Cada tipo de diagrama UML se utiliza para modelar un aspecto específico del sistema y proporciona una vista clara y comprensible de su funcionalidad y estructura.

¿Qué son los test unitarios?

Los test unitarios, son una práctica de desarrollo de software que consiste en escribir código específico para probar individualmente las unidades más pequeñas de un programa, como funciones, métodos o clases. El objetivo principal es validar que cada unidad de código funcione correctamente de manera aislada y conforme a las especificaciones.

2ª Clase (31/01/2024)



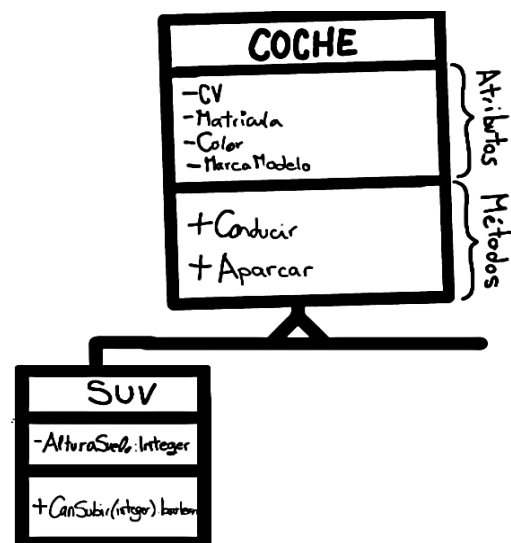
Clases: actúan como plantillas o moldes que definen las propiedades y comportamientos comunes de los objetos que se crearán a partir de ellas. Son como planos abstractos que especifican cómo se pueden crear, manipular y destruir objetos.

Instancias: (objetos), son ejemplares específicos de esas clases. Cuando se crea un objeto a partir de una clase, se está instanciando esa clase, lo que significa que se está creando una versión concreta de esa clase en tiempo de ejecución. Cada instancia tiene sus propios valores para las propiedades definidas en la clase, pero comparte los mismos métodos definidos en la clase.

-En resumen, las clases proporcionan la estructura y los comportamientos, mientras que las instancias representan los datos y las acciones específicas dentro de esa estructura.

- **Atributos:** son las características o datos que describen el estado de un objeto. Por ejemplo, si estás modelando una clase "Persona", los atributos podrían ser el nombre, la edad, la altura, etc. Los atributos representan las características que definen el estado de un objeto y están almacenados en variables dentro de la clase.

- **Métodos:** son las acciones o comportamientos que puede realizar un objeto. Estas acciones pueden incluir cualquier tipo de operación, desde cálculos matemáticos hasta la manipulación de datos. En una clase "Persona", los métodos podrían ser "caminar", "hablar", "comer", etc. Los métodos definen lo que un objeto puede hacer y están representados por funciones en la clase.



Cuando una clase "extiende" otra clase, se dice que hereda los atributos y métodos de esa clase base. La clase que se extiende se conoce como clase derivada o subclase, mientras que la clase que se extiende se llama clase base o superclase.

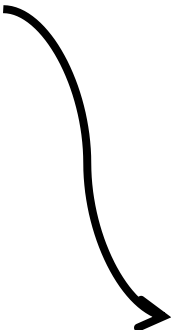
Heredar en programación orientada a objetos significa que una clase (llamada subclase o clase hija) puede adquirir propiedades y comportamientos de otra clase (llamada superclase o clase padre).

Cuando una clase hereda de otra, la subclase tiene acceso a los métodos y atributos definidos en la superclase. Esto significa que la subclase puede utilizar y extender la funcionalidad de la superclase, lo que fomenta la reutilización del código y la organización lógica de las clases.

Por ejemplo, si tienes una clase llamada "Animal" con métodos y atributos para describir animales en general, puedes crear subclases como "Perro", "Gato", "Pájaro", etc., que hereden de la clase "Animal". Esto significa que las subclases pueden heredar los métodos y atributos de la clase "Animal", como "moverse", "respirar", "nombre", "edad", entre otros, y también pueden tener sus propios métodos y atributos específicos.

Aquí otro ejemplo con la clase padre Coche y clases hijas como suv:

```
class Coche {
    Integer CV;
    String matricula;
    Integer color;
    String marcaModelo;
    Coche() {}
    /*Aquí(y arriba) puede haber public, private y protected:
    _ _ _ */ Coche (Integer CV,String matricula, Integer
color,String marcaModelo){
        this.CV=CV;
        this.matricula=matricula;
        this.color=color;
        this.marcaModelo=marcaModelo;
    }
    public void conducir() {}
    public void aparcar() {}
    protected Integer getCV(){
        return this.CV;
    }
}
```



```
class SUV extends Coche{
    Integer AlturaSuelo;
    public SUV (Integer CV,String
Matricula,Integer Color,
                String MarcaModelo, Integer
Altura){
        super(CV,Matricula,Color, MarcaModelo);
        this.AlturaSuelo=Altura;
    }
    public boolean Conducir(Integer pS){
        return pS<= this.AlturaSuelo;
    }
}
```

¿Qué es main (clase y método)?

Clase main:

La clase *main* es el punto de entrada para que se lleve a cabo la ejecución de cualquier programa Java y dentro de la clase *main*, se definen las instrucciones que se ejecutarán al comenzar. Puede estar contenida dentro de cualquier archivo Java, y su nombre no necesariamente debe ser *main*, aunque es una convención común nombrarla de esta manera.

Método *main*:

El método `main` es un método especial dentro de la clase *main*. Este método es el que se ejecuta primero cuando se inicia el programa. Ejemplo:

public static void *main* (String[] args)

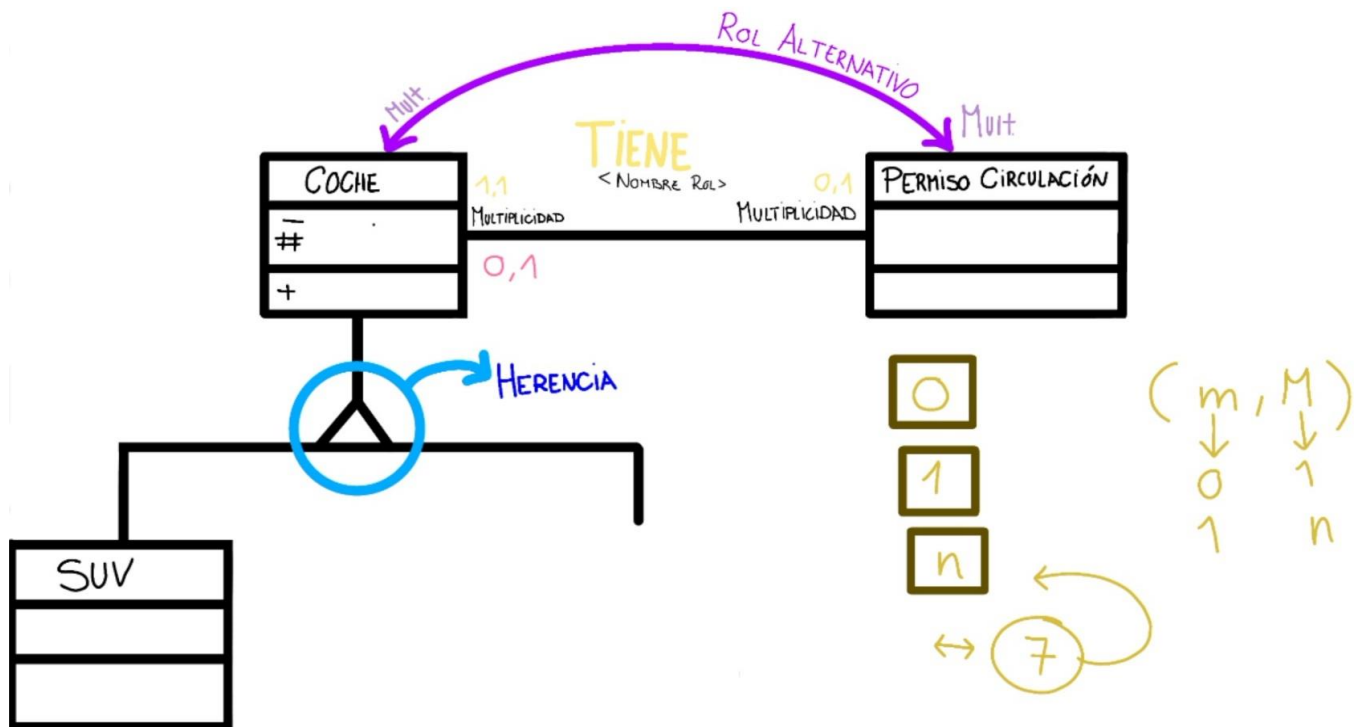
- *static*: Indica que el método *main* pertenece a la clase en sí misma, no a una instancia específica de la clase.
- *void*: Indica que el método *main* no devuelve ningún valor.
- *main*: Es el nombre del método (no necesariamente tiene que llamarse así).
- *String[] args*: Es un array de cadenas que contiene los argumentos de la línea de comandos que se pueden pasar al programa al iniciarlo.

¿Por qué `!@` necesitamos?

Son necesarios porque proporcionan un punto de entrada claro y definido para que el sistema operativo comience a ejecutar un programa Java, cuando se inicia el sistema operativo busca la clase y ejecuta su método *main*, lo que inicia la ejecución del programa. Sin este método, el programa no tendría un punto de entrada claro y no se ejecutaría correctamente.

```
public class EjemploCoche {  
    public static void main(String Arg[]) {  
        SUV micoche = new SUV(200, "9999-ZZZ", 1, "LEXUS NX450H",  
18);  
        SUV tuCoche = new SUV(900, "1111-AAAA", 2, "MERCEDES EQC",  
17);  
        if (miCoche.canSubir(22)) {  
            System.out.println("Puede subir.");  
        }  
    }  
}
```

3ª Clase (07/02/2024)



La multiplicidad se refiere a la cantidad de instancias de una clase que pueden estar asociadas con instancias de otra clase en una relación.

En los diagramas de clases de UML, la multiplicidad se representa utilizando números o rangos para indicar cuántas instancias de una clase pueden estar relacionadas con una instancia de otra clase. Esta multiplicidad se especifica en ambos extremos de la relación entre las clases.

La multiplicidad también puede ser más específica, como (0..1) para indicar que puede haber cero o una instancia relacionada, (1..5) para indicar que puede haber de uno a cinco instancias relacionadas, etc. Y es muy importante en el modelado de datos porque define las reglas y restricciones sobre cómo las instancias de las clases están conectadas entre sí en un sistema. Ayuda a definir la estructura y el comportamiento de las relaciones entre las clases, lo que facilita el diseño y la comprensión de los sistemas complejos.

A continuación diferentes ejemplos relacionadas con la multiplicidad en la P.O.O. :

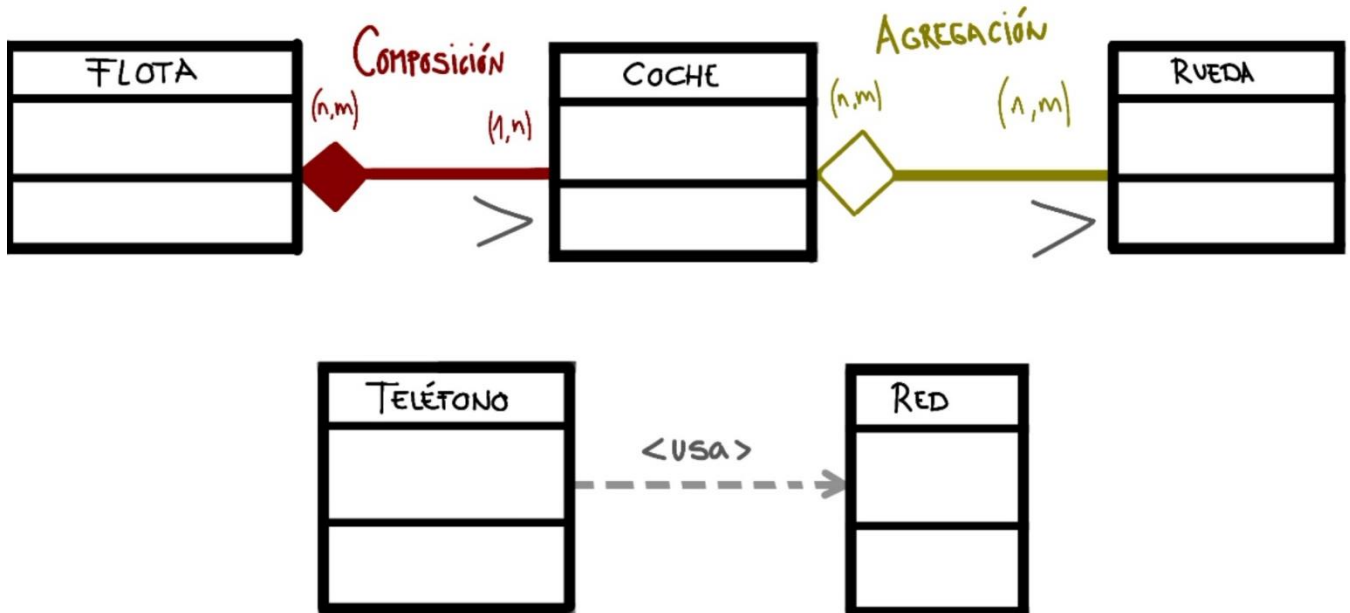
```
class Data{
    public name="Antonio";
}
```

```
class Multej01 {
    Data rol;
    Data rol=null;
}
```

```
class Multej0n {
    lista rol=new lista();
}
```

```
class Multej11 {
    Data rol=new Data();
    public Multej11(Data el){
        rol=el;
    }
}
```

```
class Multej1n {
    lista rol=new lista();
    public Multej1n(Data el){
        rol.add(el);
    }
}
```



En la composición, una clase (llamada clase contenedora) contiene directamente a otra clase (llamada clase contenida) como parte de su estructura, la clase contenida no puede existir sin la clase contenedora. Esta última es responsable de crear y gestionar la clase contenida. ((Por ejemplo, una casa contiene habitaciones. Si la casa se destruye, las habitaciones también se destruyen.))

Es una relación muy fuerte y restrictiva.

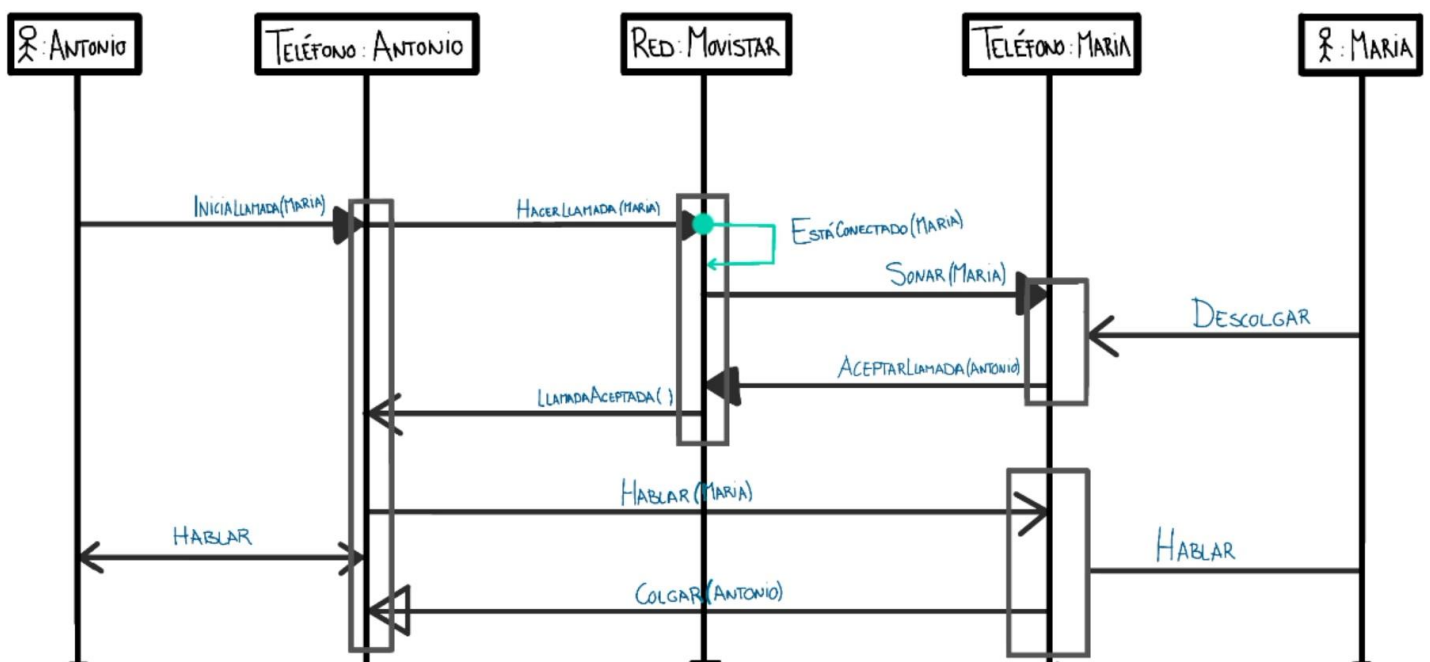
En la agregación, una clase (llamada clase contenedora) tiene una referencia a otra clase (llamada clase contenida), pero esta última puede existir de forma independiente. La clase contenedora no tiene la responsabilidad de crear o destruir la clase contenida.

((Por ejemplo, una universidad tiene una lista de estudiantes. Los estudiantes pueden existir independientemente de la universidad y pueden ser compartidos por varias universidades.))

Es una relación más débil y menos restrictiva.

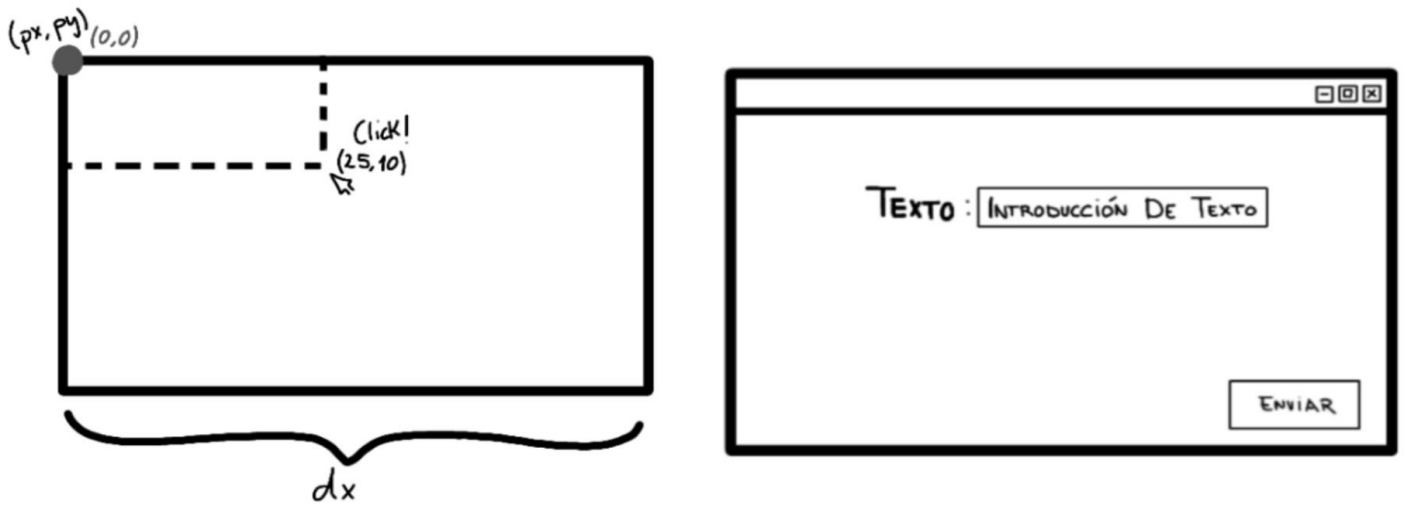
Los rectángulos verticales en gris oscuro representan la ejecución de la llamada.

Y en su conjunto representa cómo debería de funcionar el sistema.

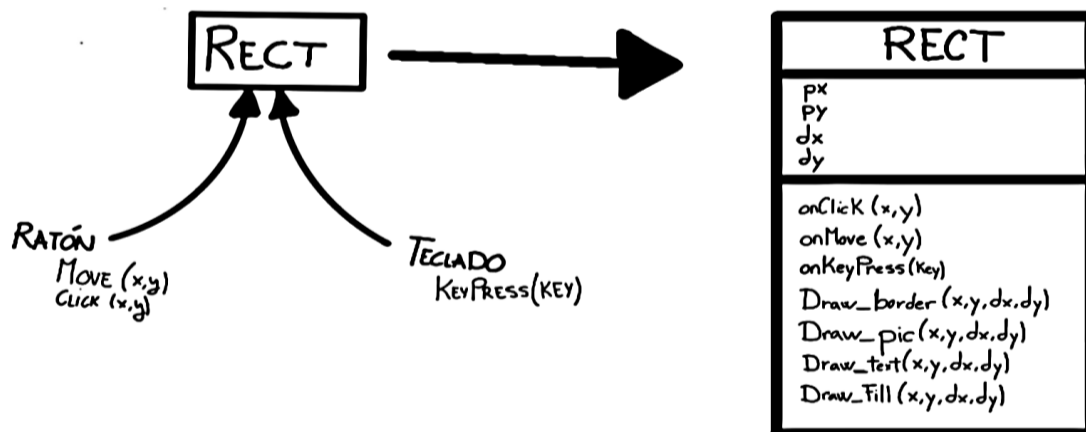


Ejercicio extra (para dentro de dos semanas):

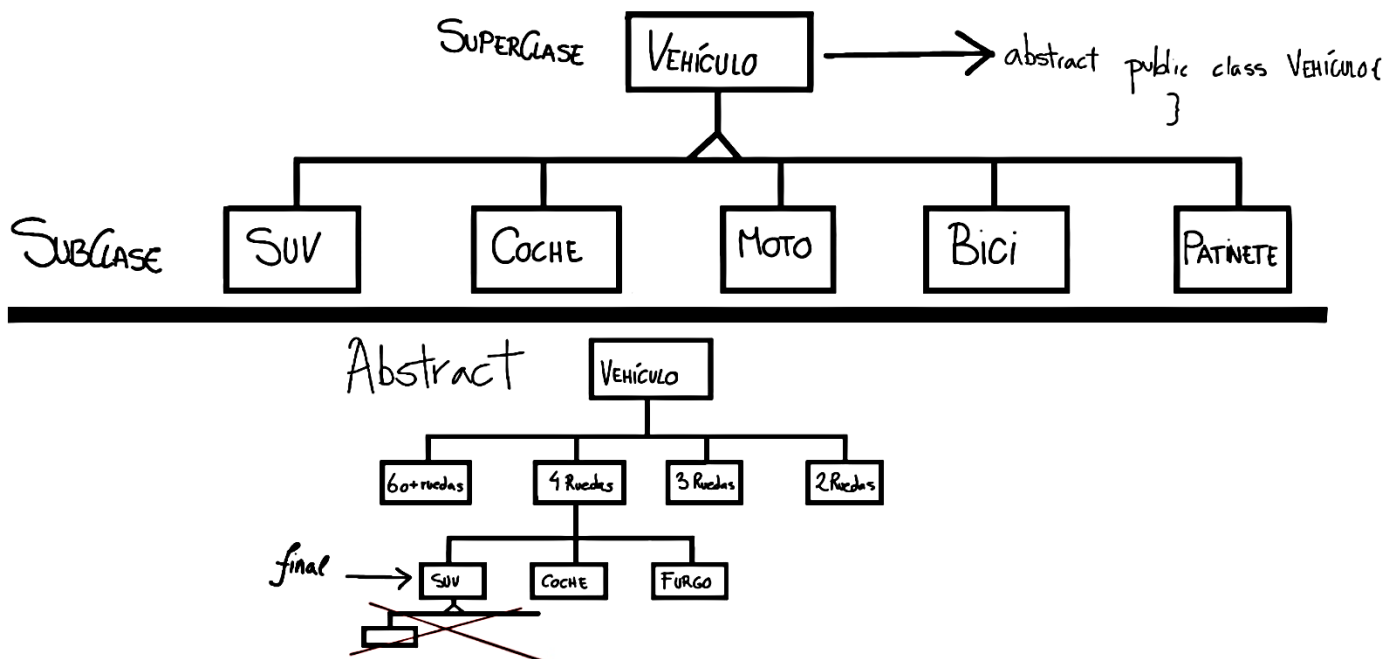
Hacer una pantalla emergente como la siguiente en plantUML (herramienta que nos permite crear diagramas UML de manera textual):



Y una pequeña pista de cómo debemos comenzar:



4ª Clase (14/02/2024)



- Superclase: es una clase que tiene características y comportamientos comunes que pueden ser compartidos por otras clases. Es la clase de la que se derivan otras y define un conjunto de atributos y métodos que son heredados por sus subclases.

- Subclases: son clases que heredan características y comportamientos de una superclase. Pueden añadir nuevos atributos y métodos, y también pueden sobrescribir o modificar los métodos heredados de la superclase, también puede tener una relación de especialización con respecto a su superclase, es decir, puede ser una versión más específica o especializada de ella.

Como ya hemos comentado antes, la herencia permite la reutilización del código y la organización jerárquica de las clases. Las subclases pueden heredar comportamientos y atributos de su superclase, lo que promueve la cohesión y la coherencia en el diseño del software.

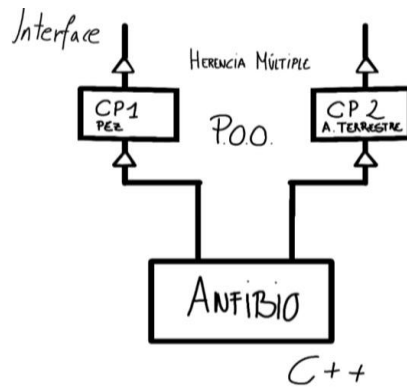
¿Qué es una clase o método abstracto?

Una clase abstracta es una clase que contiene al menos un método abstracto o que puede ser marcada como abstracta con la palabra clave `abstract`. No se pueden crear instancias (objetos) de una clase abstracta y puede contener métodos concretos (implementados) y métodos abstractos (sin implementación).

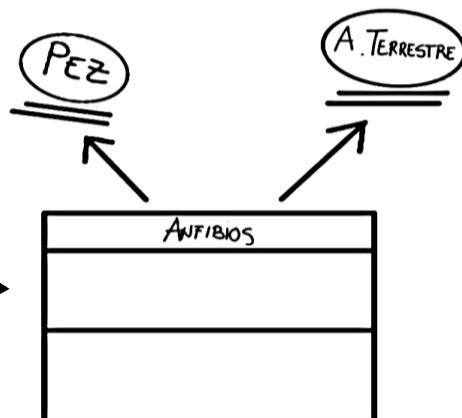
Un método abstracto es un método declarado en una clase abstracta pero no implementado en esa clase, no tiene cuerpo (es decir, no tiene llaves `{}` y código dentro de ellas) y termina con un punto y coma `;`. Los métodos abstractos deben ser implementados en las clases que heredan de la clase abstracta.

¿Qué es una interfaz?

En C++ se vería así.



Y en java se vería como algo así:



Ahora bien, ¿qué es?

Una interfaz en Java agrupa métodos abstractos y constantes, lo que facilita la implementación de la herencia múltiple, permitiendo que diversas clases compartan una misma estructura. Los métodos en una interfaz deben ser públicos (public) y se declaran sin implementación, dejando a las clases que implementan la interfaz la responsabilidad de definir su funcionalidad.

Pero entonces, ¿cuál es la diferencia entre una interfaz y una clase abstracta?

Es importante distinguir entre una interfaz y una clase abstracta en Java. Mientras que una interfaz se centra en la declaración de métodos sin implementación, una clase abstracta puede contener métodos implementados que serán heredados por subclases. Además, las clases abstractas admiten una única herencia, mientras que las interfaces permiten la implementación de múltiples interfaces.

```
package Animales;
```

```
public interface Pez {
    public String tipo();
    public String toString();
}
```

```
package Animales;
```

```
public interface AnimalTerrestre
{
    public int getNumPatras();
    public String toString();
}
```

```
package Animales;
```

```
public class Anfibio implements Pez,AnimalTerrestre {
    public String tipo() {....}
    public String toString() {....}
    public int getNumPatras() {....}
    public String toString() {....}
}
```

¿Qué es un polimorfismo?

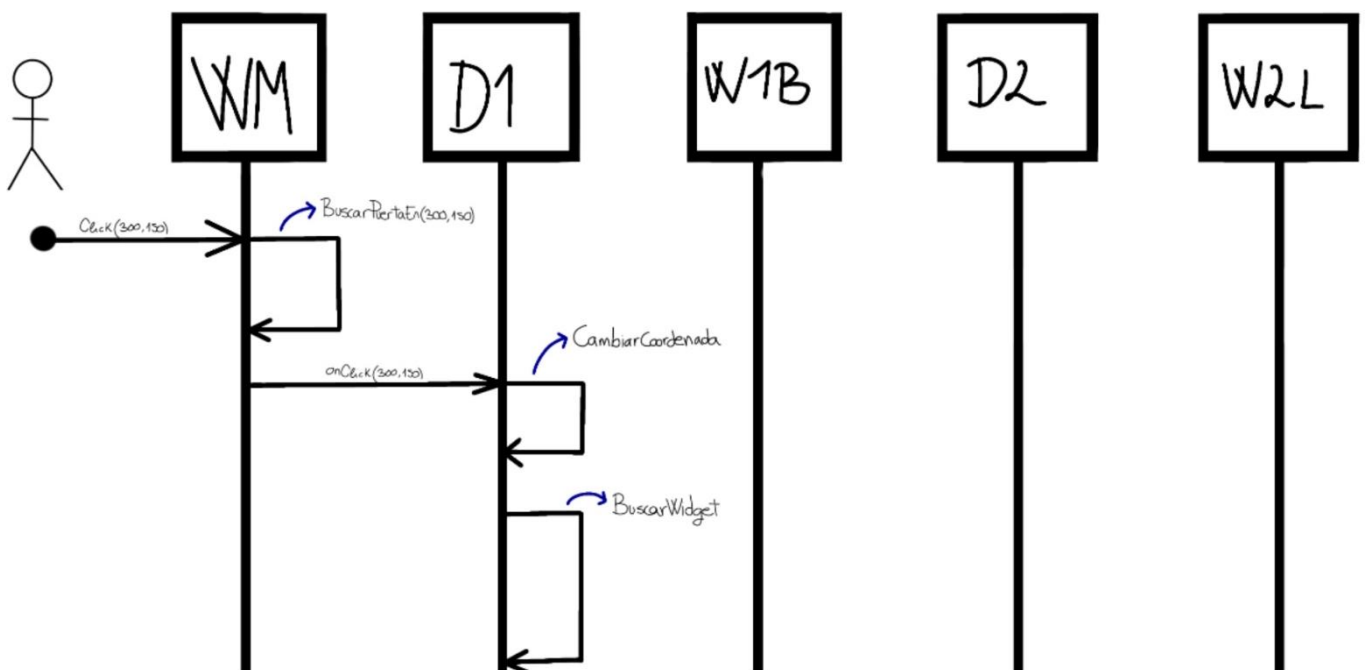
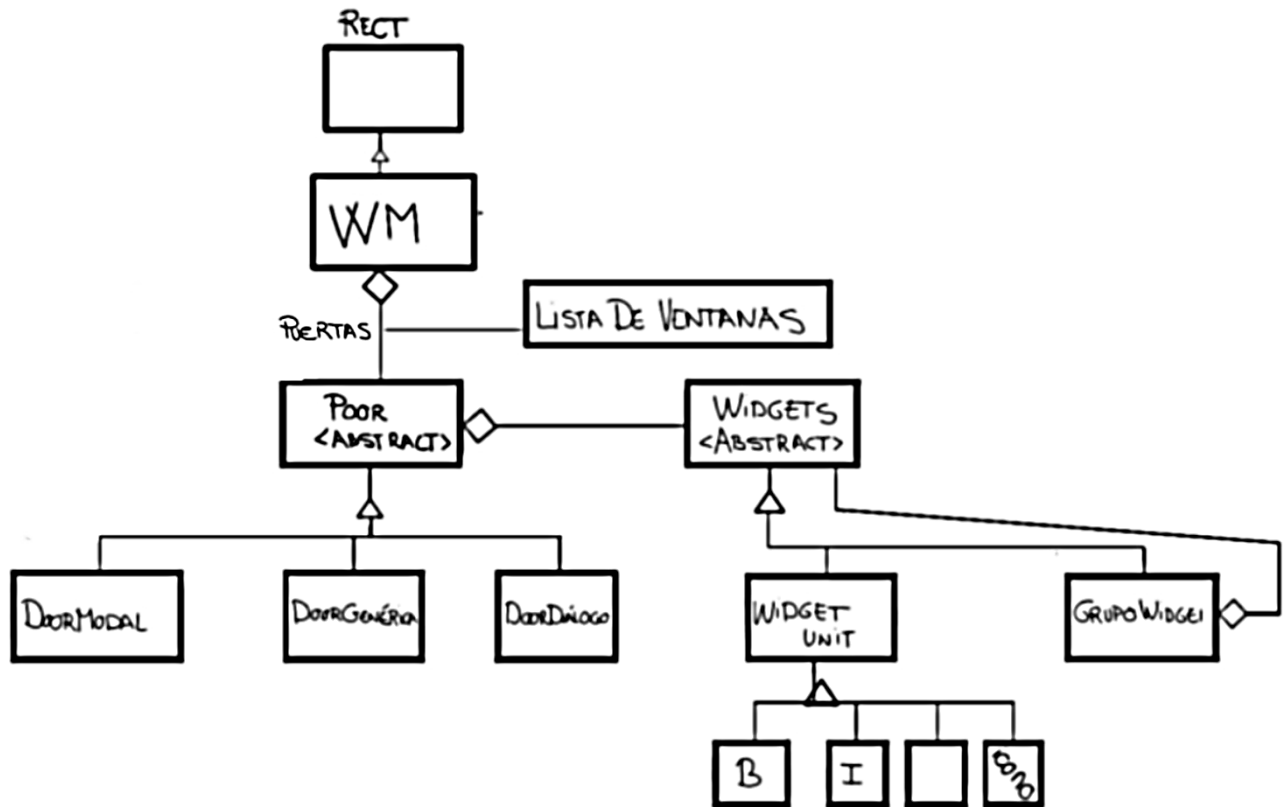
Es capacidad que tienen los objetos de una clase de comportarse de manera distinta dependiendo del contexto en el que se utilicen permite la interacción uniforme con objetos de diferentes clases a través de interfaces comunes, es esencial en la P.O.O. Hay dos tipos principales de polimorfismos:

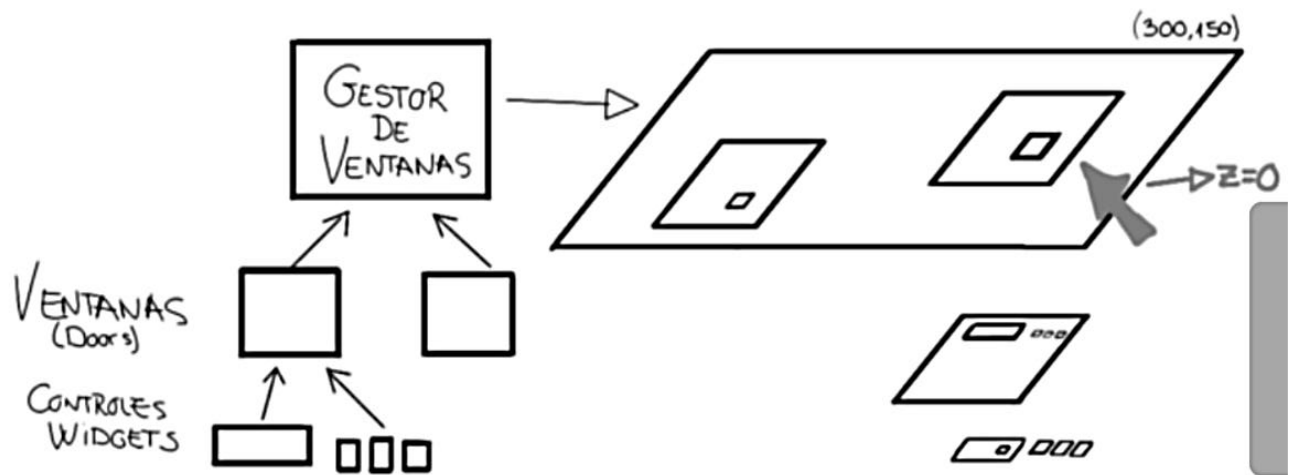
1. Polimorfismo de sobrecarga (Overloading): Este tipo de polimorfismo permite que una clase tenga varios métodos con el mismo nombre, pero con diferentes parámetros. El compilador decide qué método usar según los tipos y la cantidad de parámetros proporcionados.

2. Polimorfismo de sobreescritura (Overriding): Aquí, una subclase puede ofrecer una implementación específica de un método ya definido en una de sus superclases. Esto posibilita que un objeto de la subclase se comporte de manera distinta al llamar al mismo método definido en la superclase.

5ª Clase (14/02/2024)

La solución al ejercicio propuesto dos semanas antes es:





Día 6 (28/02/2024)

Corrección de un examen (PDF en el BlackBoard).

PREGUNTA 1

- i. d)
- ii. b)

PREGUNTA 2

****Contestada más arriba, en teoría.**

PREGUNTA 3

```
public class Cuadrado {
    double lado;
    private Cuadrado() {}
    public Cuadrado(double lado) {
        this.lado=lado;
    }
    public double getPerimetro() {
        return this.lado*4;
    }
    public double getArea() {
        return this.lado*this.lado;
    }
}
```

PREGUNTA 4

```
public abstract class Ventana {
    void mover() {
    }
}
```

```
public class Ventanuco extends Ventana {
    Bisagra[] sujeta=new Bisagra[n]();
    Integer dimension;
    Manija tirador=new Manija();
    @Override
    void mover() {
    }
    void cerrar() {
    }
}
```

PREGUNTA 5

