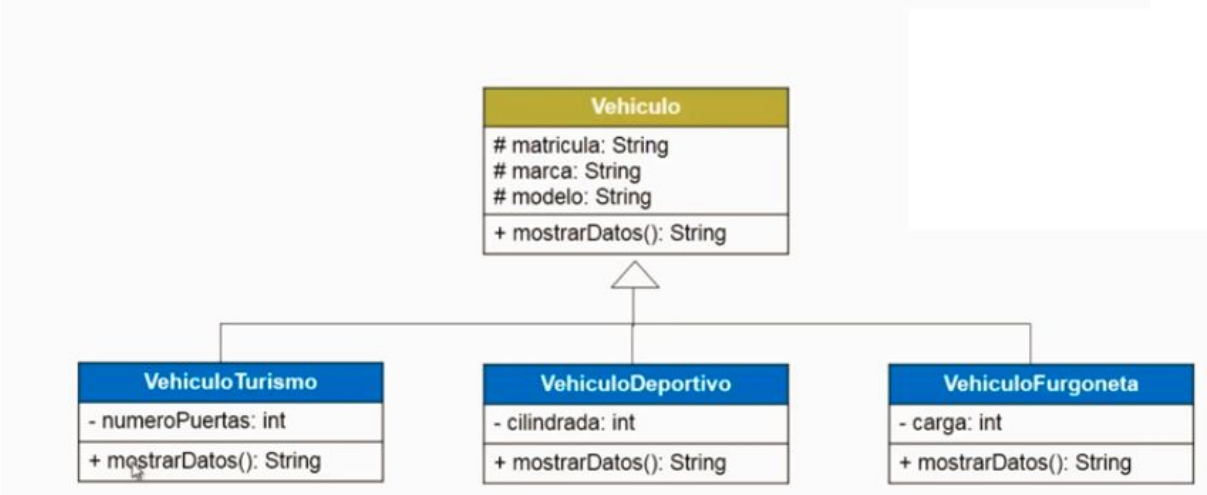
Polimorfismo

Para que exista el polimorfismo debe existir la herencia.
"Polimorfismo" es una palabra de origen griego que significa***"muchas* formas"**.
Este término se utiliza en la POO para "***referirse a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos***".

En otros términos: polimorfismo las **muchas formas** que puede tomar un objeto. Es decir, el polimorfismo consiste en conseguir que un objeto de una clase se comporte como un objeto de cualquiera de sus subclases, dependiendo de la forma de llamar a los métodos de dicha clase o subclases.

Primero veamos el siguiente diagrama de UML:



Donde podemos crear objetos del tipo Vehiculo asi:

```
Vehiculo miVEhiculo = new Vehiculo(“12gbf”, “Ferrari”, “A8”);
```

Ahora si volvemos al concepto de polimorfismo:
Polimorfismo: consiste en conseguir que un objeto de una super clase se comporte como un objeto de cualquiera de sus subclases, dependiendo de la forma de llamar a los métodos de dicha clase o subclases.

Entonces, un objeto de la clase **Vehiculo** puede almacenar cualquier objeto de su subclase: Es decir que nosotros podemos hacer:

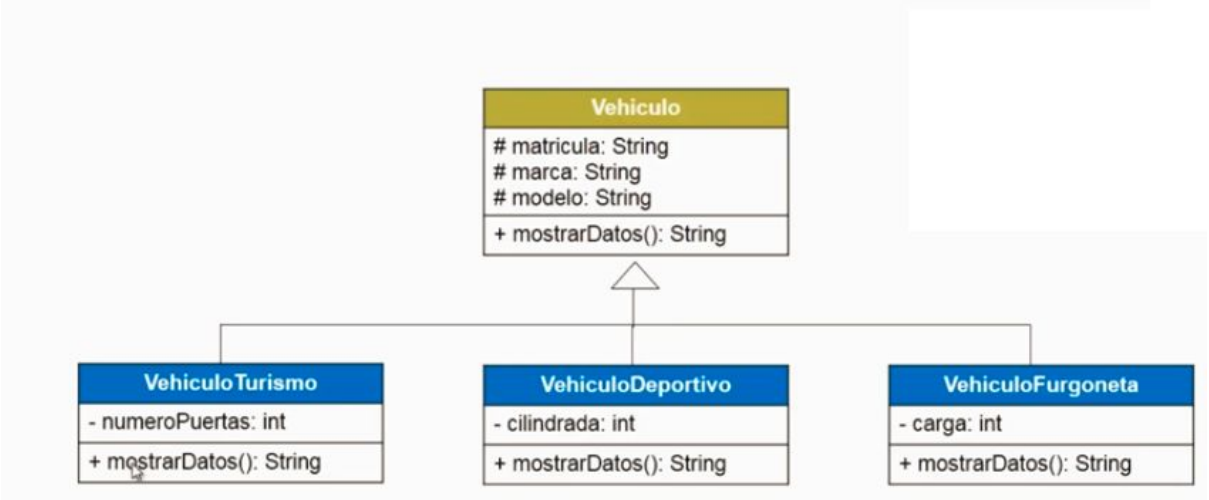
```
Vehiculo miVEhiculo2 = new VehiculoTurismo(“12gbf”, “Ferrari”, “A8”, 4);
```

Donde creamos un objeto del tipo Vehiculo y lo instanciamos por la subclase **VehiculoTurismo** y a la hora de rellenar los parámetros, tenemos que pasar tanto los atributos de la superclase **Vehiculos** y de la subclase **VehiculoTurismo**

Así con el resto, siguiendo con el concepto de polimorfismo podemos crear un objeto he instanciarlo con cualquiera de las subclases:

```
Vehiculo miVehiculo3 = new VehiculoDeportivo("12gbf", "Ferrari",
"A8", 500);
Vehiculo miVehiculo4 = new VehiculoDeportivo("12gbf", "Ferrari",
"A8", 2000);
```

Ahora vamos armar un proyecto que se llame: primerEjPolimorfismo, y creamos las clases como lo indica el diagrama de clases UML mostrado arriba:



Además creamos el constructor en la clase Vehiculo, junto con los setters y getters.

El método mostrarDatos queda en la clase Vehiculo asi:

```
public String mostrarDatos(){
    return "Matricula: "+matricula+"\nMarca: "+marca+"\nModelo: "+modelo;
}
```

Creamos la clase **VehiculoTurismo,** agregamos el constructor y generamos el getter y setter para numero de puertas y además vamos a **SOBREESCRIBIR** nuestro método: mostrarDatos()
Porque decimos que lo **sobreescribimos**, porque necesitamos crear un método propio pero que cuando se llame desde un objeto del tipo **VehiculoTurismo ese método muestre un atributo adicional: el número de puertas.**

Cuando decimos que sobreescribimos un método en este caso el método mostrarDatos() usamos la palabra reservada **@Override** para indicarle al compilador de java que estamos sobreescribiendo dicho método.

Lo mismo hacemos con el resto de las clases del diagrama:
VehiculoDeportivo VehiculoFurgoneta

Ahora vamos a crear una clase Principal donde agregamos nuestro método main:

Ahora usando un for each vamos a ver cómo los diferentes objetos actúan según como han sido instanciados:

Observemos que obtenemos por salida, donde vamos a ver los diferentes mensajes que obtenemos según el objeto que instanciamos usando el polimorfismo.