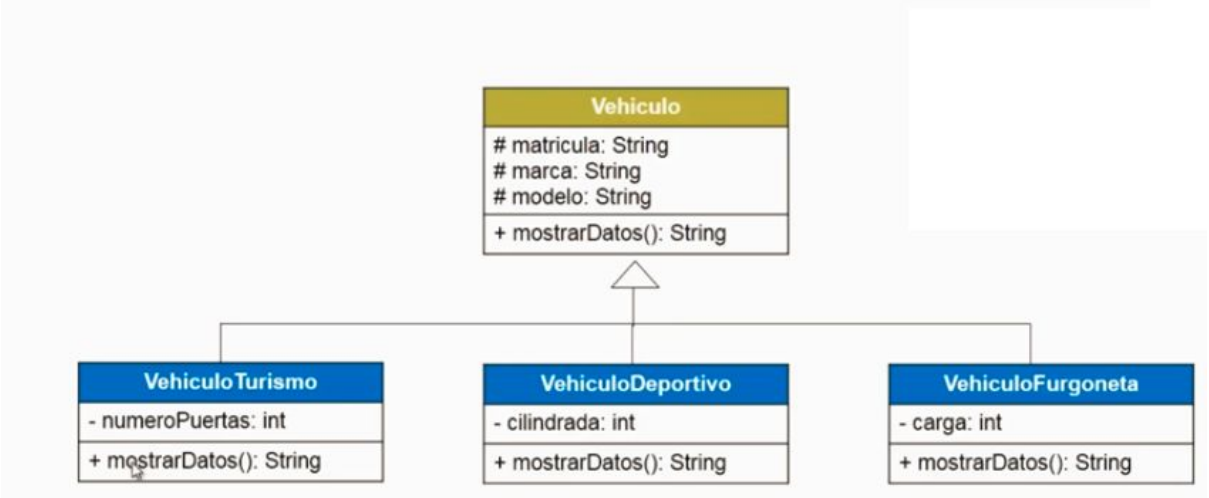
Polimorfismo

Importante: 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 en detalle el siguiente diagrama de UML:



En el Diagrama vemos que existe una relación de Herencia entre las clases,y es correcto si creamos un objeto del tipo Vehiculo asi:

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

Ahora volviendo al concepto de polimorfismo:

Polimorfismo: consiste en conseguir que un objeto de una superclase 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, podemos usar polimorfismo y crear un objeto de la clase **Vehiculo que pueda** 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 de nombre `miVehiculo2` y lo instanciamos con la subclase **VehiculoTurismo.**
Además debemos tener presente que a la hora de rellenar los parámetros, tenemos que pasar tanto los atributos de la superclase **Vehiculos** y de la subclase **VehiculoTurismo, por eso pasamos el 4.**

Siguiendo con el concepto de polimorfismo podemos crear dos objetos más, por ejemplo, he instanciarlos con cualquiera de las subclases:

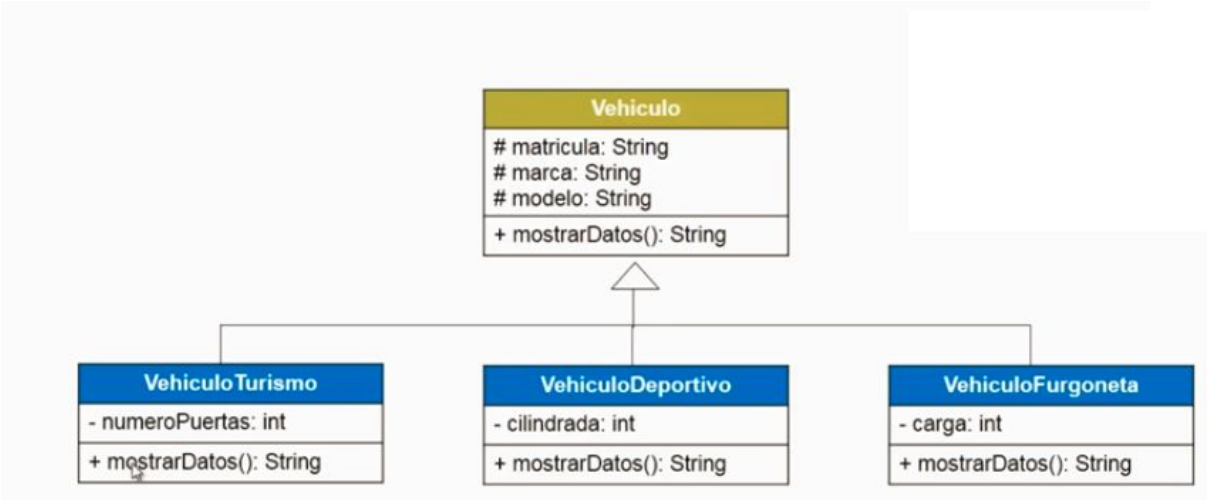
```
Vehiculo miVehiculo3 = new VehiculoDeportivo("12gbf", "Ferrari",
"A8", 500);
Vehiculo miVehiculo4 = new VehiculoDeportivo("12gbf", "Ferrari",
"A8", 2000);
```

Vamos a practicar el polimorfismo, siguiendo los siguientes pasos:

1- Crear un proyecto que se llame: **primerEjPolimorfismo**.
2- Creamos 5 clases: **Vehiculo, VehiculoTurismo, VehiculoDeportivo, VehiculoFurgoneta** y por último una clase llamada **Principal**.

***Nota**:Recorda que los nombres de las clases empiezan con mayúscula!*

3- En la Clase **Vehiculo** creamos el constructor de la clase Vehiculo, junto con los setters y getters. según el siguiente diagrama de clases UML mostrado arriba:



4- Generamos un método de nombre mostrarDatos(), en la clase Vehiculo:

5- Creamos la clase **VehiculoTurismo** la hacemos que herede de la clase Vehiculo, y agregamos el constructor, 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.

@Override: Cada vez que se tiene una clase que hereda un método de una superclase, se tiene la oportunidad de sobreescribir el método (a menos que dicho método esté marcado como *final*). El beneficio clave al sobreescribir un método heredado es la habilidad

de **definir un comportamiento específico** para los objetos de la subclase.
El método que se sobreescribe va en la subclase y se le coloca la notación @Override arriba de la firma del mismo.

Las reglas básicas para la sobreescritura de métodos son las siguientes:

+ La lista de argumentos del método debe ser exactamente la misma.
+ El tipo de retorno debe de ser el mismo o un subtipo del tipo de retorno declarado originalmente.
+ El nivel de acceso **no debe de ser más restrictivo**.
+ El nivel de acceso **puede ser menos restrictivo**.
+ Los métodos de instancia pueden ser sobreescritos solamente si han sido heredados por la subclase.
+ Los métodos sobreescritos pueden arrojar cualquier excepción no verificada(de tiempo de ejecución) por el compilador.
+ Los métodos sobreescritos **NO** pueden arrojar excepciones verificadas por el compilador.
+ No se puede sobreescibir un método marcado como **final**.
+ No se puede sobreescibir un método marcado como **estático** (**static**).
+ Si un método no puede ser heredado, no puede ser sobreescrito.

Invocar la versión de la superclase de un método sobreescrito:

En algunas ocasiones necesitamos invocar al método escrito en la superclase en lugar de la versión que hemos sobreescrito, para ello se utiliza la palabra super seguida de un punto(.) y posteriormente el nombre del método a invocar, por ejemplo:

super.mostrarDatos();

6- Continuando con nuestra práctica, completamos el resto de las clases VehiculoDeportivo VehiculoFurgoneta haciendo lo mismo con el resto de las clases del diagrama, generamos los atributos correspondiente según el diagrama de clases dado, escribimos el constructor, los getter y los setters y el método: mostrarDatos() sobreescrito, pero recorda adaptarlo según los atributos de cada una de las clases.que indica el diagrama de clases.

7 -En la clase **Principal** agregamos nuestro método main: y usamos el polimorfismo!!!!,

8- Crear un Array de dimensión igual a 4 con el nombre **misVehiculos**, esto lo hacemos con la siguiente línea:

```
Vehiculo misVehiculos[] = new Vehiculo[4];
// creo un array de la clase Vehiculo llamado misVehiculos[] de dimension 4.
```

8- En el array que hemos creado vamos a guardar los diferentes objetos de la superclase Vehiculo para que se comporten como un objeto de cualquiera de sus subclases (polimorfismo), es decir:

```
Vehiculo misVehiculos[0] = new Vehiculo("GH67", "Ferrari", "A89");
// creo un objeto de la clase Vehiculo

Vehiculo misVehiculos[1] = new VehiculoTurismo(4,"&*HJ", "Audi", "P14");
// creo un objeto del tipo Vehiculo, lo instanciamos con la subclase
// VehiculoTurismo  y lo guardo en la posicion 1 de mi array de nombre
// misVehiculos[1].

Vehiculo misVehiculos[2] = new VehiculoDeportivo(500,"45GH", "Toyota",
"KJ8");
// creo un objeto del tipo Vehiculo, lo instanciamos con la subclase
// VehiculoDeportivo  y lo guardo en la posicion 2 de mi array de nombre
// misVehiculos.

Vehiculo misVehiculos[3] = new VehiculoFurgoneta(2000,"JI8", "Toyota", "J9");
// creo un objeto del tipo Vehiculo, lo instanciamos con la subclase
// VehiculoFurgoneta  y lo guardo en la posicion 3 de mi array de nombre
// misVehiculos.
```

Es decir estamos usando polimorfismo para generar un objeto de la superclase Vehiculo y este se comporte como un objeto de cualquiera de sus subclases.

9 - Armamos un for each para ver cómo los diferentes objetos actúan según como han sido instanciados:

10- Compilamos y ejecutamos nuestro código: Observemos que muestra la salida de nuestro programa, vamos a ver los diferentes mensajes que se obtiene según el objeto que instanciamos.

Preguntas?????

Práctica de Polimorfismo:

1- Abrimos el proyecto practicaDeHerencia,

2- En la clase padre creamos un método según la siguiente notación de UML:

+ mostrarMiProposito(): void

3- En el cuerpo del método creado, agregamos un showMessageDialog que diga: "Yo soy tu Padre"

4- En la clase Cliente sobreescribimos el método + mostrarMiProposito(): void para que diga en un showMessageDialog "Yo estoy destinado a comprar y pagar!!!"

5- En la clase Administrador sobreescribimos el método + mostrarMiProposito(): void para que diga en un showMessageDialog "Yo solo soy un Administrador Feliz!!!"

6- En la Clase principal creamos un método según la siguiente notación de UML:

+ mostrarEjemploPolimorfismo(): void

7- En el cuerpo del método creado, vamos a crear un array de nombre: objetoPolimorfico de dimensión igual a 3.

Nota: creamos una array no un arrayList!

8- Guardamos en la posición 0 de nuestro array, un objeto del tipo Persona.
En la posición 1 de nuestro array almacenamos un objeto del tipo Persona instanciando un Cliente.
En la posición 2 de nuestro array creado ingresamos un objeto del tipo Persona, instanciando un Administrador.

9- Armamos un foreach para que nos ejecute o llame a mostrarMiProposito() según el objeto creado.

10 - En el método main invoco al método mostrarEjemploPolimorfismo().

11 - Compilo, ejecuto y analizo la salida del programa.

12 - Excelente trabajo!!!! ya sabes como se sobreescribe un método y como se usa el polimorfismo en Java.