Resumen de guía 4

Java

Conceptos básicos

- Herencia
  - Visibilidad
  - Constructores
  - Sobreescritura
- Polimorfismo
- Interfaces

Herencia

Es el mecanismo mediante el cual una clase es capaz de heredar todas las características (atributos y métodos) de otra clase, permitiendo la reutilización de código.

Los atributos y métodos comunes se definen en la **superclase** y las subclases heredan los mismos. Además la **subclase** puede tener atributos y métodos propios, que también puede sobreescribir.

La manera de usar herencia es a través de la palabra ***extends***.

Constructores. **super**

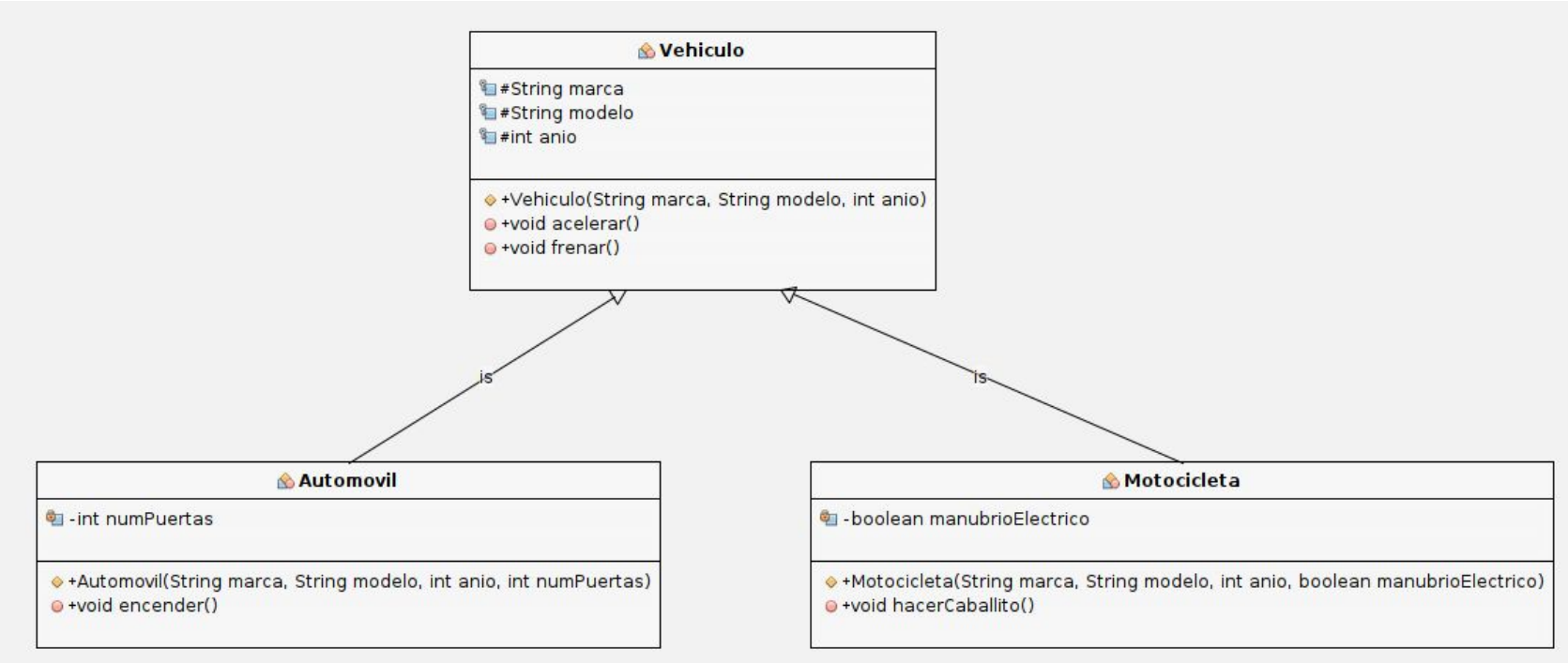
Una diferencia entre los constructores y los métodos es que los constructores no se heredan, pero los métodos sí. Todos los constructores definidos en una superclase pueden ser usados desde constructores de las subclases a través de la palabra clave super. La palabra clave super es la que me permite elegir qué constructor, entre los que tiene definida la clase padre, es el que debo usar.

La palabra clave **super** nos sirve para hacer referencia o llamar a los atributos, métodos y constructores de la **superclase** en las **subclases**.

Visibilidad a través de la herencia

Visibilidad	Public	Private	Protected	Default
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	NO	SI	SI
Desde una Subclase del mismo Paquete	SI	NO	SI	SI
Desde una Subclase fuera del mismo Paquete	SI	NO	SI, a través de la herencia	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

Ejemplo


Vehiculo
#String marca
#String modelo
#int anio
+Vehiculo(String marca, String modelo, int anio)
+void acelerar()
+void frenar()
is
is
Automovil
-int numPuertas
+Automovil(String marca, String modelo, int anio, int numPuertas)
+void encender()
Motocicleta
-boolean manubrioElectrico
+Motocicleta(String marca, String modelo, int anio, boolean manubrioElectrico)
+void hacerCaballito()

Ejemplo

```java
public class Vehiculo {
    protected String marca;
    protected String modelo;
    protected int anio;

    public Vehiculo(String marca, String modelo, int anio) {
        this.marca = marca;
        this.modelo = modelo;
        this.anio = anio;
    }

    public void acelerar() {
        System.out.println(x: "El vehículo acelera");
    }

    public void frenar() {
        System.out.println(x: "El vehículo frena");
    }
}
```

```java
public class Motocicleta extends Vehiculo {
    private boolean manubrioElectrico;

    public Motocicleta(String marca, String modelo, int anio, boolean manubrioElectrico) {
        super(marca, modelo, anio);
        this.manubrioElectrico = manubrioElectrico;
    }
    @Override
    public void acelerar() {
        System.out.println(x: "La motocicleta acelera velozmente");
    }

    public void hacerCaballito() {
        System.out.println(x: "La motocicleta hace un caballito");
    }
}
```

```java
class Automovil extends Vehiculo {
    private int numPuertas;
    public Automovil(String marca, String modelo, int anio, int numPuertas) {
        super(marca, modelo, anio);
        this.numPuertas = numPuertas;

    }
    @Override
    public void acelerar() {
        System.out.println(x: "El automovil acelera rápidamente");

    }
    public void encender() {
        System.out.println(x: "El automóvil se enciende");

    }
}
```

Sobreescritura *@Override*

La **sobreescritura** de métodos permite a las subclases proporcionar una implementación específica de un método heredado de su superclase, adaptándolo a sus necesidades particulares. Esto promueve la flexibilidad y la reutilización de código en el desarrollo de software.

Se utiliza cuando queremos cambiar o personalizar el comportamiento de un método heredado de la superclase en la subclase. **Para lograrlo, la subclase redefine el método utilizando la misma firma (nombre, parámetros y tipo de retorno, o covariante) que el método en la superclase.**

La sobreescritura es posible gracias a la relación de herencia entre clases. Cuando una subclase redefine un método de la superclase, se utiliza la anotación **@Override**

Polimorfismo

El **polimorfismo** se refiere a la capacidad de un objeto de una clase específica para ser tratado como un objeto de una clase más general, lo que permite que **distintos objetos respondan de manera diferente a una misma invocación de método.**

El polimorfismo se basa en la **herencia** y la **sobreescritura** de métodos. El polimorfismo permite que una variable de tipo de la clase base pueda referirse a un objeto de cualquiera de sus subclases.

```
Vehiculo vehiculo1 = new Automovil(marca: "Toyota", modelo: "Corolla", anio: 2022, numPuertas: 4);
Vehiculo vehiculo2 = new Motocicleta(marca: "Honda", modelo: "CBR600RR", anio: 2021, manubrioElectrico: true);

vehiculo1.acelerar();  // El método acelerar de Automovil se ejecuta
vehiculo2.acelerar();  // El método acelerar de Motocicleta se ejecuta
```

Interfaces

Una **interfaz** es sintácticamente similar a una clase abstracta, en la que puede especificar uno o más métodos que no tienen cuerpo. Esos métodos deben ser implementados por una clase para que se definan sus acciones.

Una i**nterface especifica qué se debe hacer, pero no cómo hacerlo.** Una vez que se define una interface, cualquier cantidad de clases puede implementarla. Además, una clase puede implementar cualquier cantidad de interfaces.

Ejemplo de implementación de interfaces

```java
package interfaces;
public abstract class Figura {
    abstract double calcularArea();
    abstract double calcularPerimetro();
}
```

```java
public interface FiguraTridimensional {
    double calcularVolumen();
}
```

```java
public interface FiguraBidimensional{
    public FiguraBidimensional clonar();
}
```

```java
public class Circulo extends Figura implements FiguraTridimensional, FiguraBidimensional{
    private double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI * radio * radio;
    }

    @Override
    public double calcularPerimetro() {
        return 2 * Math.PI * radio;
    }

    @Override
    public FiguraBidimensional clonar() {
        return new Circulo(radio);
    }

    @Override
    public double calcularVolumen() {
        return (3/4*Math.PI*radio*radio*radio);
    }
```