Polimorfismo:

Polimorfismo significa “muitas formas”. Em Orientação a Objetos, o conceito do polimorfismo é aplicado quando utilizamos o vertbo SER entre pelo menos 2 ou mais subclasses, podendo ser feito utilizando-se interfaces ou Classes abstratas.

Na programação orientada a objetos, o **polimorfismo** permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Assim, é possível tratar vários tipos de maneira homogênea (através da interface do tipo mais abstrato). O termo ***polimorfismo*** é originário do grego e significa "muitas formas" (*poli* = muitas, *morphos* = formas).

O polimorfismo é caracterizado quando duas ou mais classes distintas tem métodos de mesmo nome, de forma que uma função possa utilizar um objeto de qualquer uma das classes polimórficas, sem necessidade de tratar de forma diferenciada conforme a classe do objeto.[1]

Uma das formas de implementar o polimorfismo é através de uma classe abstrata, cujos métodos são declarados mas não são definidos, e através de classes que herdam os métodos desta classe abstrata.[2]

Polimorfismo utilizando Classes Abstratas:

Outra maneira de representarmos o polimorfismo é através do uso de classes abstratas. Uma Classe abstrata é uma Classe que pode conter atributos, métodos e construtores como uma Classe comum, mas que também pode ter métodos definidos como abstratos, similares aos da interface.

Regras sobre classes abstratas:

- Uma Classe abstrata pode conter métodos abstratos.
- Para que possamos declarar um método como abstrato, a Classe deve ser do tipo abstrata.
- Quando uma Classe comum herda de uma Classe abstrata, ela é obrigada, assim como no caso da interface, a implementar os métodos.

```
package entity;

public abstract class Automovel {

    private Integer idAutomovel;
    private String nome;
```

```
	public Automovel() {
	}

	public Automovel(Integer idAutomovel, String nome) {
		this.idAutomovel = idAutomovel;
		this.nome = nome;
	}

	@Override
	public String toString() {
		return idAutomovel + ", " + nome;
	}

	public Integer getIdAutomovel() {
		return idAutomovel;
	}
	public void setIdAutomovel(Integer idAutomovel) {
		this.idAutomovel = idAutomovel;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}

	// Métodos abstratos
	public abstract void setFabricante(String fabricante);
	public abstract String getFabricante();

}
```

```
package entity;

public class CarroEsportivo extends Automovel{

	private Integer ano;
	private String fabricante;

	public CarroEsportivo() {
```

```
	}

	public CarroEsportivo(Integer idAutomovel, String nome, Integer
ano, String fabricante) {
		super(idAutomovel, nome);
		this.ano = ano;
		this.fabricante = fabricante;
	}

	@Override
	public String toString() {
		return super.toString() + ", " + ano + ", " + fabricante;
	}

	public Integer getAno() {
		return ano;
	}
	public void setAno(Integer ano) {
		this.ano = ano;
	}

	@Override
	public String getFabricante() {
		return fabricante;
	}

	@Override
	public void setFabricante(String fabricante) {
		this.fabricante = fabricante;
	}
}
```

```
package entity;

public class CarroExecutivo extends
Automovel{

	private String modelo;
	private String fabricante;

	public CarroExecutivo() {

	}
```

```
	public CarroExecutivo(Integer idAutomovel, String nome, String
modelo, tring fabricante) {
		super(idAutomovel, nome);
		this.modelo = modelo;
		this.fabricante = fabricante;
	}

	@Override
	public String toString() {
		return super.toString() + ", " + modelo + ", " + fabricante;
	}

	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}

	@Override
	public String getFabricante() {
		return fabricante;
	}

	@Override
	public void setFabricante(String fabricante) {
		this.fabricante = fabricante;
	}
}
```

```
package main;

import entity.Automovel;
import entity.CarroEsportivo;
import entity.CarroExecutivo;

public class Main2 {

	public static void main(String[] args) {

		Automovel a1 = new CarroEsportivo(1, "Ferrari", 2012,
"Ferrari Italia");
```

```
            Automovel a2 = new CarroExecutivo(2, "C4", "Sedan",
"Citroen");
            System.out.println(a1);
            System.out.println(a2);
      }
}
```

No console...

```
1, Ferrari, 2012, Ferrari Italia
2, C4, Sedan, Citroen
```

No exemplo acima podemos dizer que CarroEsportivo É Automovel e CarroExecutivo É Automovel, portanto, a herança da Classe abstrata configura o uso de Polimorfismo. Note que Transformamos o objeto Automovel em CarroEsportivo e CarroExecutivo