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 Interfaces:

A interface é o tipo de programação mais “puro” do Java, pois não programamos o conteúdo dos métodos de uma interface, apenas sua declaração (assinatura). Toda interface Java obedece às seguintes regras:

- Todos os métodos de uma interface são implicitamente públicos e abstratos.
- Todos os métodos de uma interface não possuem corpo, apenas assinatura.
- Os Atributos de uma interface são, por definição, constantes, ou seja, possuem valor final.
- Quando uma Classe implementa uma interface, a Classe deverá forecer corpo para todos os métodos da interface, exceto se a Classe for abstrata.
- Uma interface pode herdar de outras interfaces.
- Uma classe pode implementar várias interfaces.

Tipos de polimorfismo:

Existem quatro tipos de polimorfismo que a linguagem pode ter (atente para o fato de que nem toda linguagem orientada a objeto tem implementado todos os tipos de polimorfismo):

- **Universal**
  - Inclusão - um ponteiro para classe mãe pode apontar para uma instância de uma classe filha (exemplo em Java: `List lista = new LinkedList();` (tipo de polimorfismo mais básico que existe)
  - Paramétrico - se restringe ao uso de templates (C++, por exemplo) e generics (C#/Java)

- **Ad-Hoc**
  - Sobrecarga - duas funções/métodos com o mesmo nome mas assinaturas diferentes
  - Coerção - conversão implícita de tipos sobre os parâmetros de uma função

---

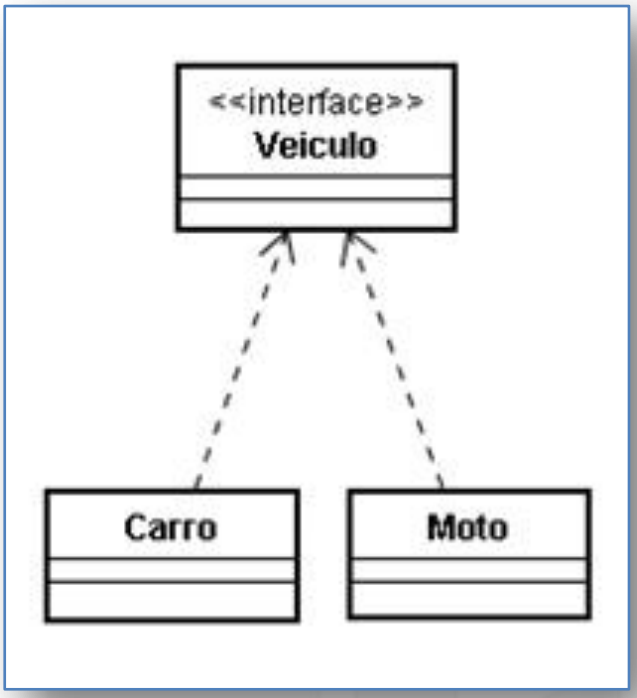
```
package entity;

public interface Veiculo {

	public void setPlaca(String placa);
	public String getPlaca();
}
```

```
package entity;

public class Carro implements Veiculo{

	private Integer idCarro;
	private String nome;
	private String placa;

	public Carro() {

	}

	public Carro(Integer idCarro, String nome, String placa) {
		this.idCarro = idCarro;
		this.nome = nome;
		this.placa = placa;
	}
```

```
	@Override
	public String toString() {
		return idCarro + ", " + nome + ", " + placa;
	}

	public Integer getIdCarro() {
		return idCarro;
	}
	public void setIdCarro(Integer idCarro) {
		this.idCarro = idCarro;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}

	@Override
	public String getPlaca() {
		return placa;
	}

	@Override
	public void setPlaca(String placa) {
		this.placa = placa;
	}
}
```

```
package entity;

public class Moto implements Veiculo{

	private Integer idMoto;
	private String modelo;
	private String placa;

	public Moto() {

	}

	public Moto(Integer idMoto, String modelo, String placa) {
```

```
		this.idMoto = idMoto;
		this.modelo = modelo;
		this.placa = placa;
	}

	@Override
	public String toString() {
		return idMoto + ", " + modelo + ", " + placa;
	}

	public Integer getIdMoto() {
		return idMoto;
	}
	public void setIdMoto(Integer idMoto) {
		this.idMoto = idMoto;
	}
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}

	@Override
	public String getPlaca() {
		return placa;
	}

	@Override
	public void setPlaca(String placa) {
		this.placa = placa;
	}

}
```



No modelo podemos dizer que Carro É Veículo e Moto É Veículo, portanto, a implementação de interface é um relacionamento do tipo É-UM. A Vantagem deste tipo de abordagem é o uso do Polimorfismo, pois podemos "Transformar" a Interface Veículo passando para ela uma instância de Carro ou Moto

---

```
package main;

import entity.Carro;
import entity.Moto;
import entity.Veiculo;

public class Main {

	public static void main(String[] args) {
		//Polimorfismo

				Veiculo v1 = new Carro(1, "Ferrari", "ABC-1234");
				Veiculo v2 = new Moto(2, "Suzuki", "ABC-4321");

				System.out.println(v1);
```

```
                System.out.println(v2);
        }
}
```

No console...

```
1, Ferrari, ABC-1234
2, Suzuki, ABC-4321
```