COTI Informática

Revisão: Interfaces, Classes Abstratas e Polimorfismo

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.

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.

Exemplo de uma interface Veiculo

```
package entity;

public interface Veiculo {

   public void setPlaca(String placa);

   public String getPlaca();

}
```

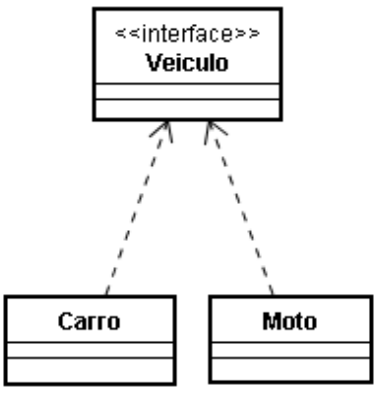
Classes implementando a interface Veiculo

```
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;
	}
}
```



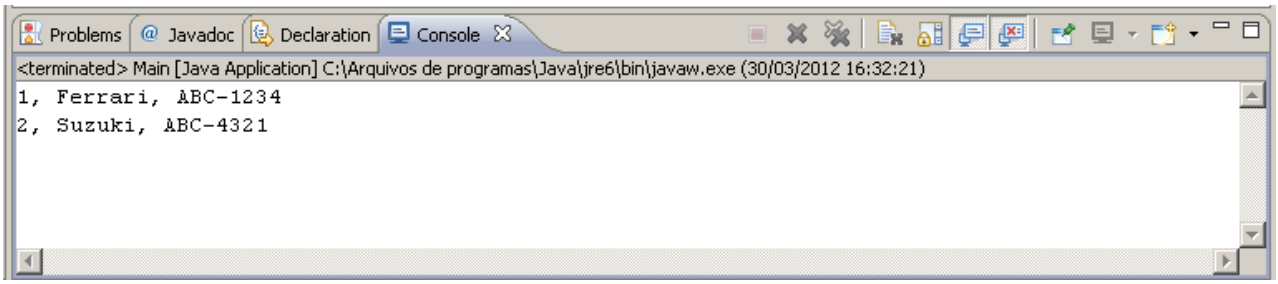
No modelo acima podemos dizer que **Carro É Veículo** e **Moto É Veículo**, portanto, a implementação de interface é um relacionamento do tipo **É-UM**

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**

Executando...

```
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);
	}
}
```

Resultado...



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

COTI Informática

Revisão: Interfaces, Classes Abstratas e Polimorfismo

Exemplo de uma Classe abstrata Automovel

```
package entity2;

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();
}
```

Classes comuns herdando da Classe Abstrata

```
package entity2;

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 entity2;

public class CarroExecutivo extends Automovel {

	private String modelo;
	private String fabricante;

	public CarroExecutivo() {

	}

	public CarroExecutivo(Integer idAutomovel, String nome, String modelo,
			String 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;
	}
}
```

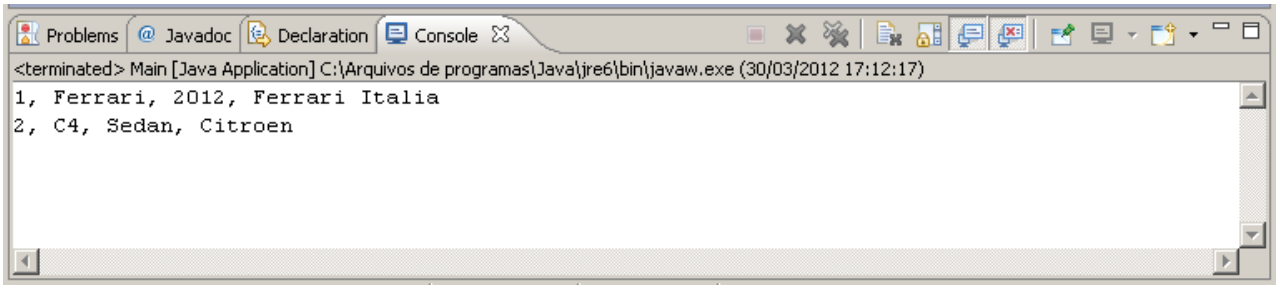
Executando...

```
package main;

import entity2.Automovel;
import entity2.CarroEsportivo;
import entity2.CarroExecutivo;

public class Main {

	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);

	}
}
```

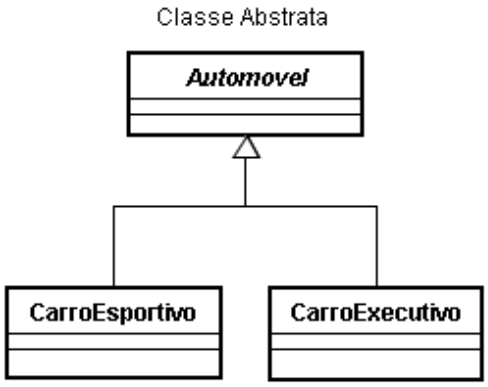
COTI Informática

Revisão: Interfaces, Classes Abstratas e Polimorfismo

Resultado...



No exemplo acima podemos dizer que **CarroEsportivo É Automovel** e **CarroExecutivo É Automovel**, portanto, a herança da Classe abstrata configura o uso de Plimorfismo



Note que Transformamos o objeto Automovel em CarroEsportivo e CarroExecutivo:

```
Automovel a1 = new CarroEsportivo();
Automovel a2 = new CarroExecutivo();
```

Exercícios

1. Crie uma interface Funcionario contendo os métodos void setFuncao(String funcao) e String getFuncao().

2. Crie duas Classes: Estagiario e Gerente que implementem a interface Funcionario. Teste na Main a execução do Polimorfismo

3. Crie uma Classe abstrata Usuario contendo idUsuario, nome e métodos abstratos setLogin(String login) e String getLogin().

4. Crie as Classes Cliente e Administrador herdando da Classe abstrata. Teste na Main o Polimorfismo