



Bicicleta caloi = new **Bicicleta**();

Orientação a Objetos

- `super()` e `this()`;
 - `super` – classe 'mãe'
 - `this` – classe atual;
 - Exemplo:

```
void mudarX(int x){  
    this.x = x;  
}
```

Orientação a Objetos

- Pacotes (introdução)

- Exemplo

```
package aula3;  
class Computador {  
    ...  
}
```

Orientação a Objetos

- O comando **package** é sempre o primeiro comando de um arquivo... ele vem antes de qualquer coisa!
- Para que a sua Classe dentro de um pacote enxergue outra classe em outro pacote, use o comando **import**

Orientação a Objetos

- `import nomeDoPacote.NomeDaClasse;`
- `import nomeDoPacote.*;`
 - Importa todas as classes de um pacote

Orientação a Objetos

- Pra que eu limite o acesso a variaveis e metodos existem modificadores de acesso!
- Java possui 4 modificadores de acesso...

public

protected

default

private

Orientação a Objetos

- Modificadores de Acesso

Classes

Só podem ser “default” ou public ..

Construtores

- private | protected

Variáveis

- Qualquer modificador...

Variáveis locais

- Não tem sentido modificadores...

Orientação a Objetos

- 'static'
 - São variáveis associadas a uma Classe e existem antes de qualquer objeto existir...
 - Variáveis static são comuns a todos os objetos de uma mesma Classe!]

ex. contagem de uma população...

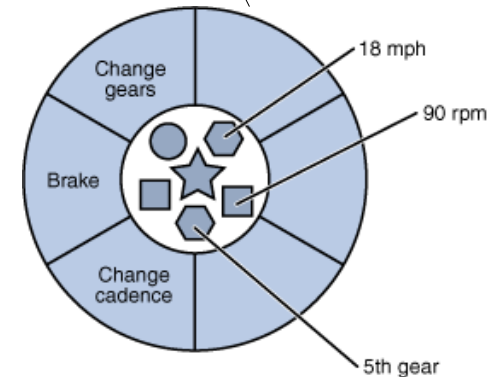
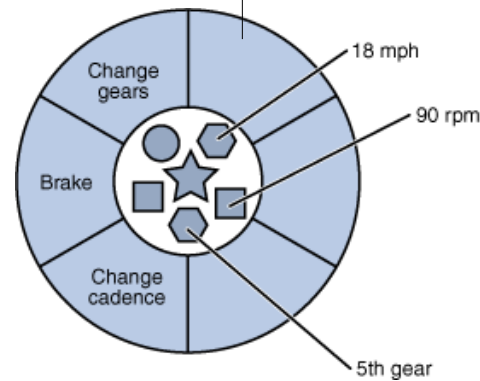
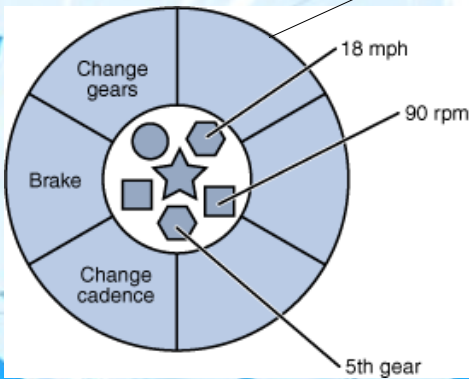
Orientação a Objetos

- 'static'
 - Podem ser chamados antes da instanciação de qualquer objeto pois estão associados a uma Classe

```
NomeDaClasse.variavelStatic;  
NomeDaClasse.metodoStatic();
```

Orientação a Objetos

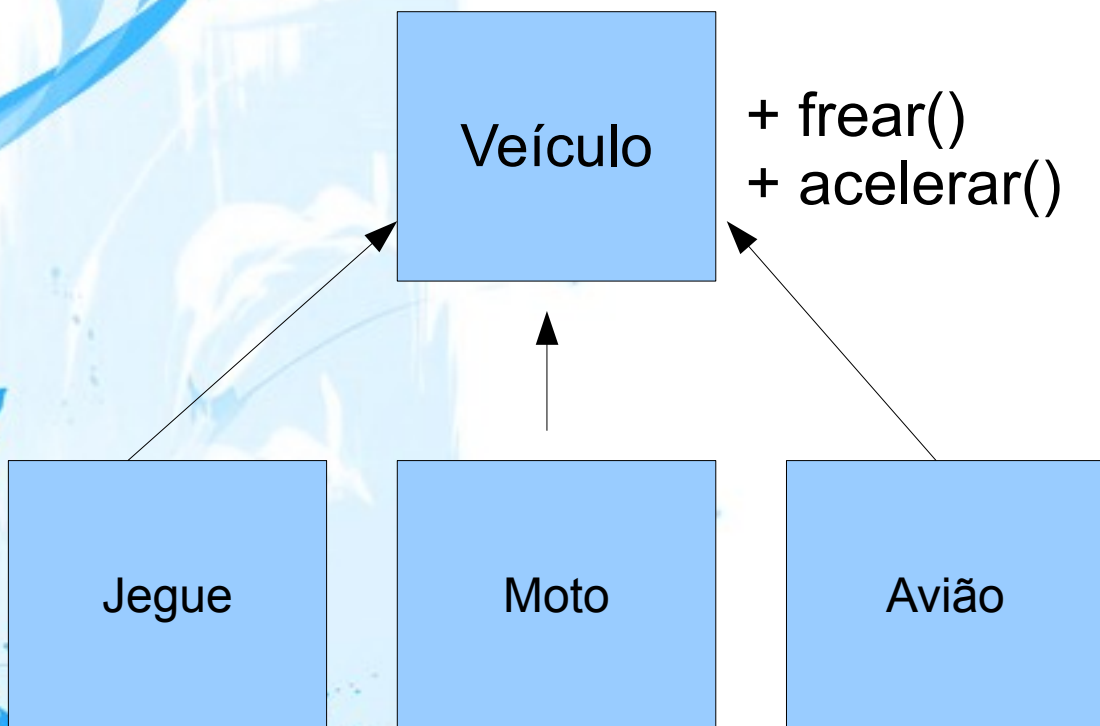
Espaço **Static**
comum a todos da classe
Bicicleta



Orientação a Objetos

- Polimorfismo – capacidade de assumir formas diferentes...
 - > veiculo > jegue,moto,aviao,...

Orientação a Objetos



```
public class Veiculo {  
  
    public void frear(){  
        System.out.println("Veiculo frear...");  
    }  
  
    public void acelerar(){  
        System.out.println("Veiculo acelerar...");  
    }  
}
```

Orientação a Objetos

Motorista

+ estacionar(Veiculo v);

```
public class Motorista {  
  
    public void estacionar(Veiculo v){  
        v.frear();  
    }  
  
}
```



Veículo

+ frear()
+ acelerar()

Jegue

Moto

Avião

Orientação a Objetos



Motorista

+ estacionar(Veiculo v);

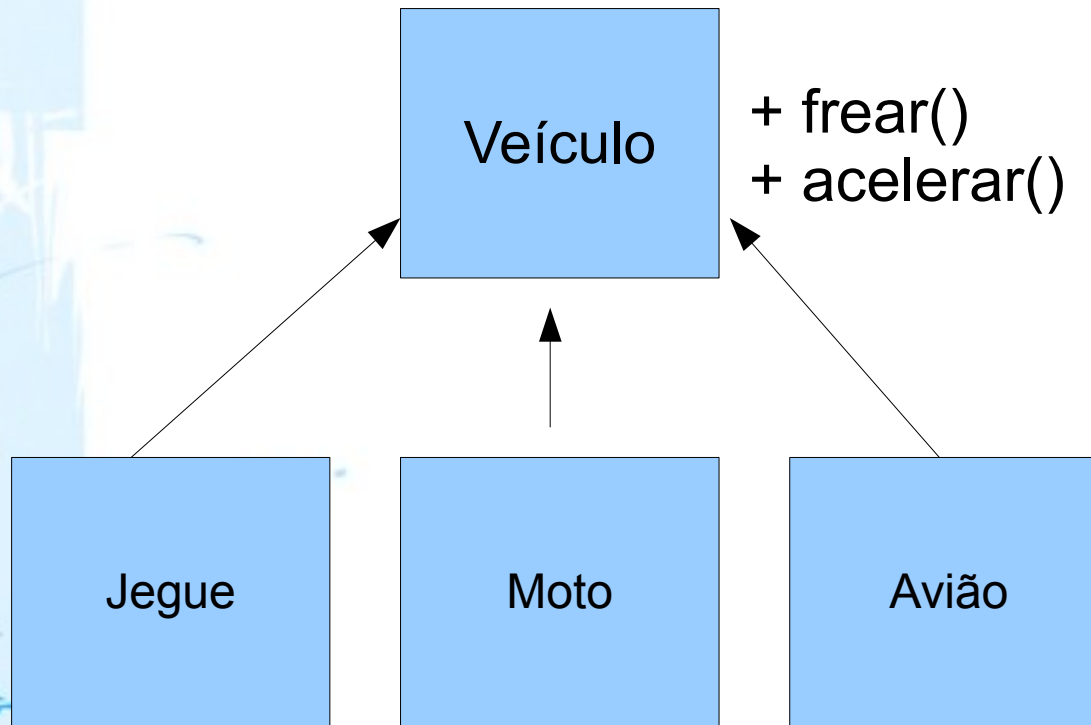
Motorista consegue estacionar qualquer Veículo de tal forma que eu não preciso alterar o código da Classe Motorista...

ou seja estacionar comporta-se de **formas diferentes** para cada tipo de Veículo

Orientação a Objetos

```
public class Principal {  
  
    public static void main(String args[]){  
  
        Motorista joao = new Motorista();  
  
        Aviao boeing = new Aviao();  
  
        joao.estacionar(boeing);  
        // como Aviao é um Veículo joao vai estacionar!  
  
    }  
}
```

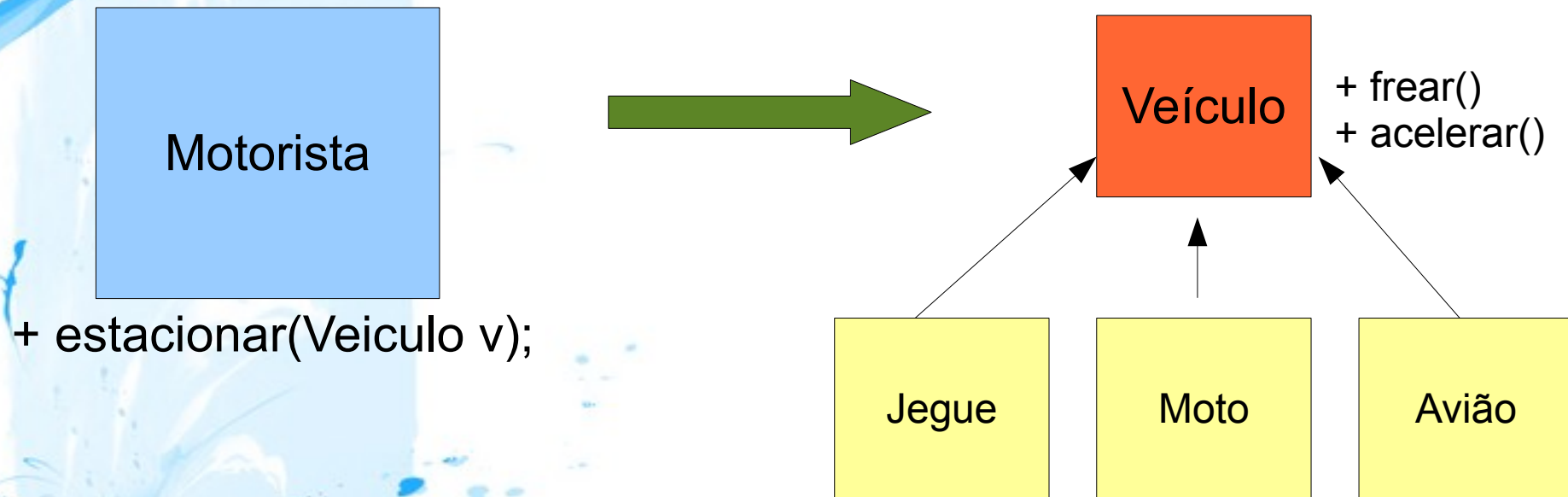
Orientação a Objetos



Orientação a Objetos

- Polimorfismo
 - Usar um objeto no lugar de outro...
 - Um objeto intermediário que só fornece as informações necessárias ...
 - ex. Motorista estacionando Veiculo!

Orientação a Objetos



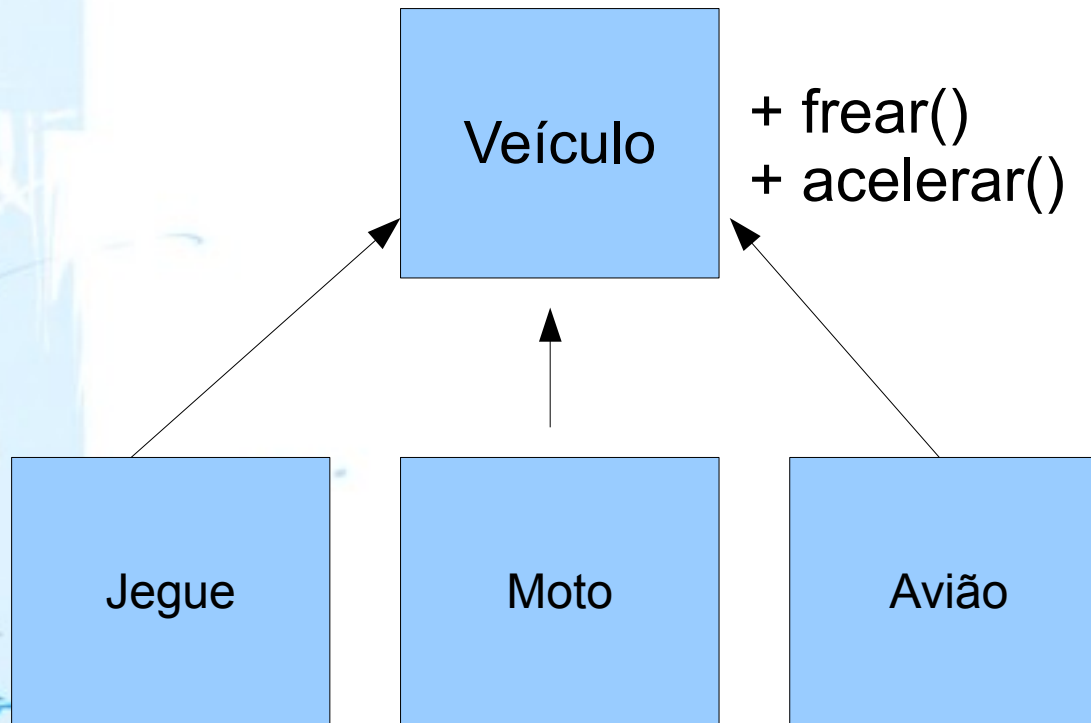
Orientação a Objetos

- Veiculo serve para que tenhamos um **polimorfismo por herança!**

Orientação a Objetos

- Mas Veiculo é um conceito muito **abstrato**...
 - **Tem sentido definir como um veiculo generico freia?**
 - Como garantir que as subclasses vão implementar **freia()** ?

Orientação a Objetos



Orientação a Objetos

- E se não houver subclasses de Veiculo?
 - Um Veiculo não pode ser construído porque ele é apenas um conceito...
 - Mas ele é ótimo como uma abstração, um intermediário, uma interface de comunicação!

Orientação a Objetos

- Dizemos que Veiculo é uma classe **abstrata** pois possui comportamentos que são **abstratos**, que não podem ser claramente definidos...
- São comportamentos que vão ser definidos nas subclasses...

Orientação a Objetos

- Então se uma classe for abstrata (**apenas um conceito**) ela não pode ser construída, ou seja não é possível criar um objeto dessa Classe...
 - caso a classe tenha **no mínimo um comportamento abstrato** ela não pode ser construída por completo...
 - **Ex:** Veículo, Mamífero, Sistema, Máquina

Orientação a Objetos

- `public abstract class NomeDaClasse { ... }`
 - O modificador **abstract** diz se uma classe é abstrata ou não em java..
 - Classes abstratas **NÃO** podem ser instanciadas, **NÃO** podem ser construídas...

Orientação a Objetos

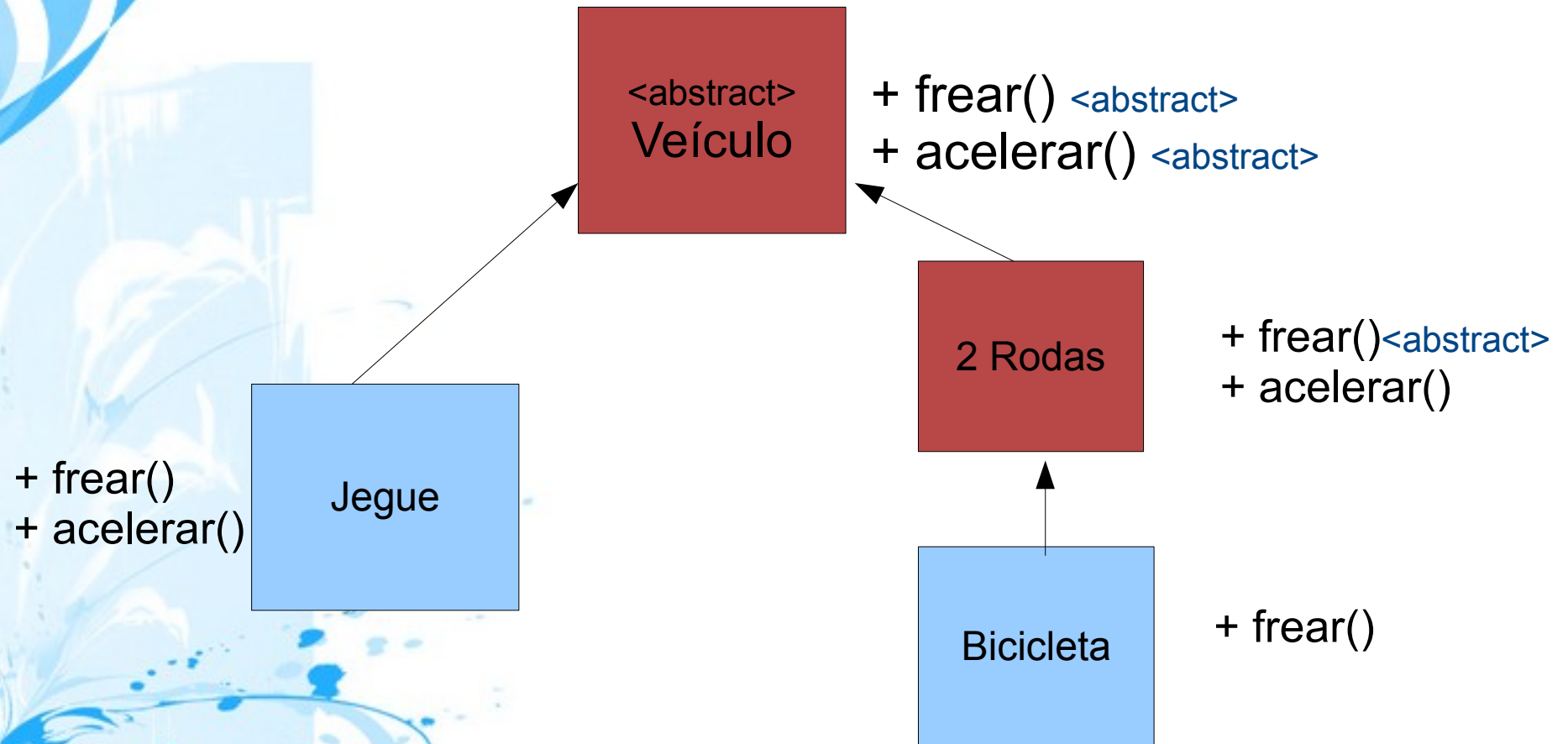
- Classes só são abstratas porque definem comportamentos abstratos.
- Para definir comportamentos abstratos usamos também o modificador **abstract** em métodos...

```
public void abstract frear();
```

Orientação a Objetos

- Classes abstratas obrigam os seus **comportamentos** a serem definidos em algum ponto da hierarquia antes de que ela possa ser instanciada... **o que?**

Orientação a Objetos



Para que **Jegue** possa ser criada ela precisa obrigatoriamente definir **frear()** e **acelerar()** caso contrário ela terá de ser **abstract** também!

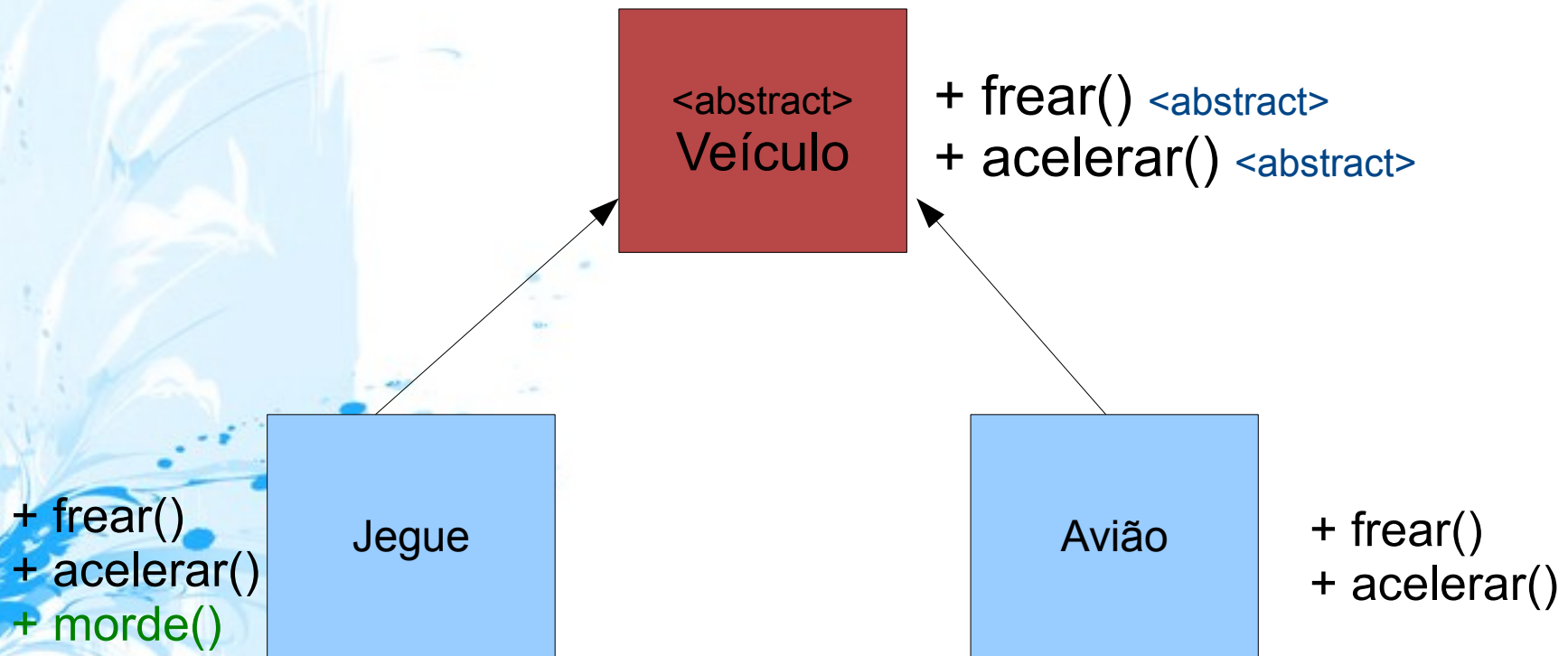
Orientação a Objetos

- Classes abstratas
 - Uma **classe abstrata pura** não deveria ter construtor pois não pode ser instanciada,
 - mas caso uma subclasse queira fazer uma referência a um construtor **super(...)**; logo elas podem ter construtores, mas não são consideradas como sendo **puras!**

Orientação a Objetos

Herança Pura vs. Extendida

- Jegue morde mas Avião não...



Orientação a Objetos

- **Interface** é uma classe abstrata “**pura**”
 - Não tem construtor
 - Só tem métodos abstratos
 - Variáveis são todas constantes...

Orientação a Objetos

- Sendo uma **interface** uma **classe abstrata pura** ela não possui
 - Construtor
 - Métodos definidos

Orientação a Objetos

```
public interface contasAritmeticas {  
    int somar(int a, int b);  
    int multiplicar(int b, int c);  
}
```

todos os métodos são considerados como sendo **public abstract** implicitamente. Porque?

Orientação a Objetos

- Podemos dizer que uma **interface** é uma espécie de contrato entre 2 classes, onde uma classe se compromete a implementar(definir) os comportamentos de uma interface

Orientação a Objetos

- Em Java como vimos todo mundo só tem uma mãe, logo só pode **herdar de uma classe**, mas uma classe pode “assinar vários contratos” ou seja **implementar várias interfaces!**

Orientação a Objetos

```
class Calculos extends Math implements  
ContasAritmeticas, ContasLogaritmicas,  
ContasComplexas {  
  
}
```

Orientação a Objetos

- Com uma interface herdando outra ela se anexa ao seu contrato tudo que está no outro...

```
public interface A extends B, C, D {  
}
```

note que B C e D também são interfaces!

Orientação a Objetos

- Quando uma classe **implementa** uma interface então dizemos que esta classe possui agora a especificação dessa interface

```
public class A implements B { ... }
```

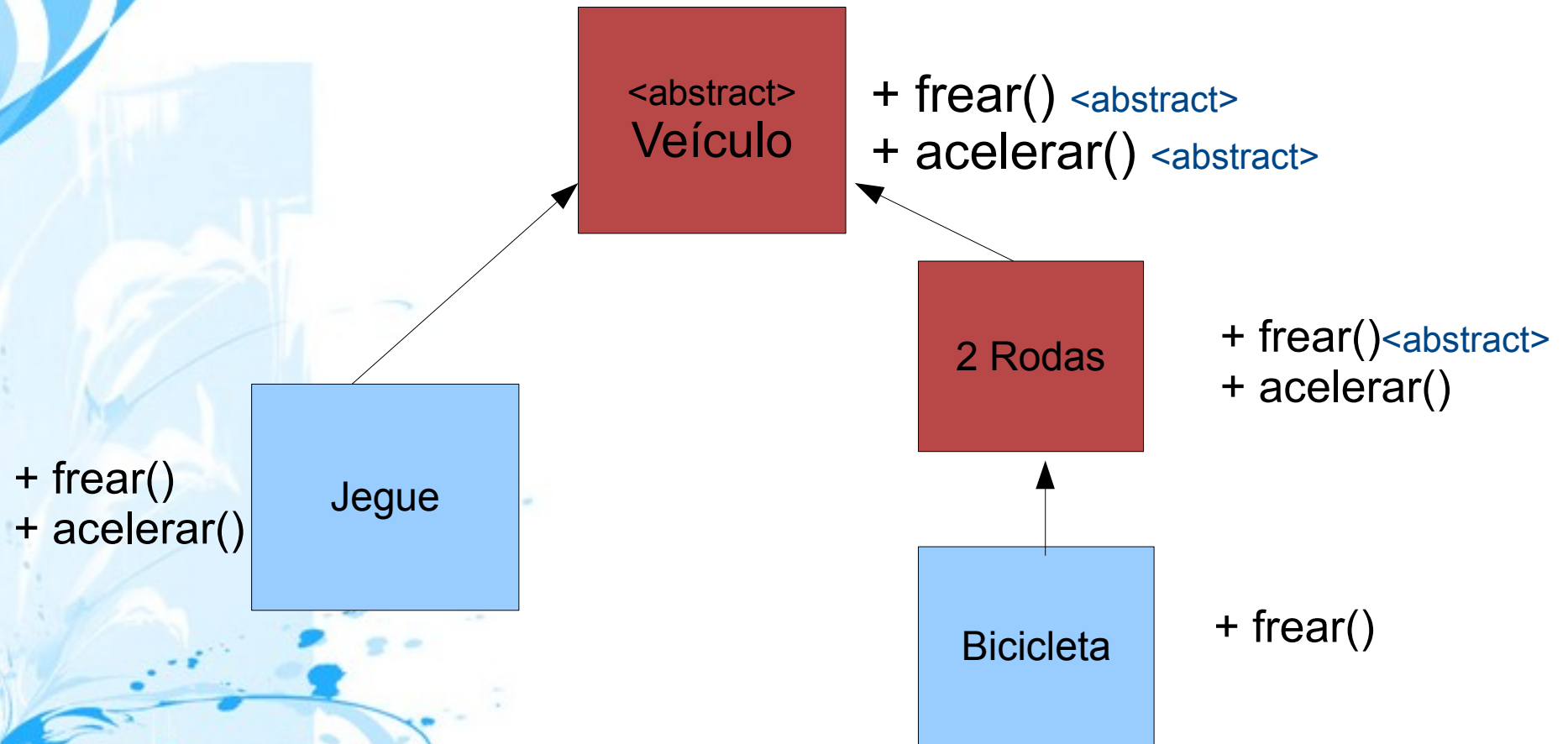
```
B obj = new A();
```

mas note que **obj** só enxergará o que estiver definido em **B**!

Orientação a Objetos

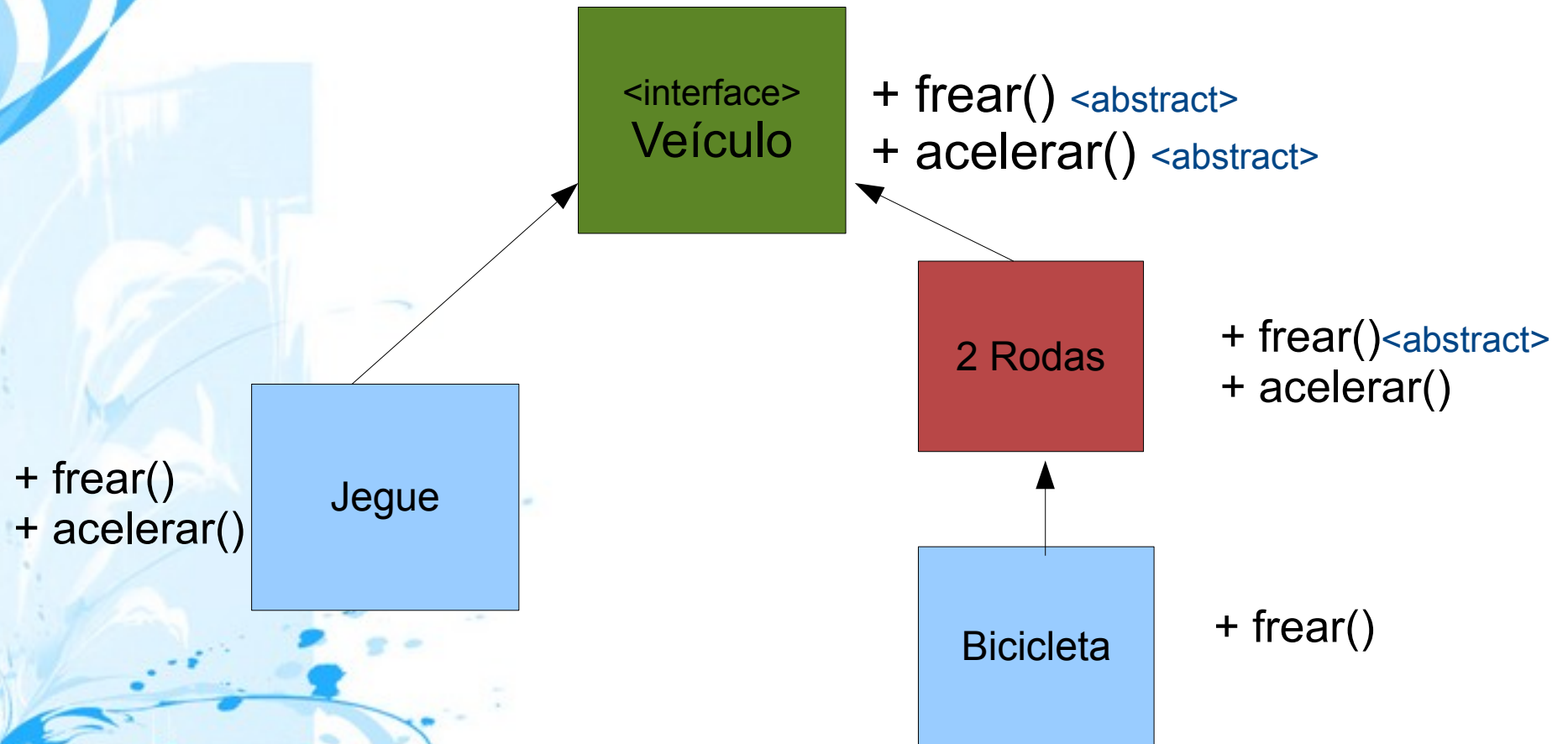
- Dizemos que interfaces fornecem **polimorfismo sem herança**
- Se eu sei que um objeto implementa uma interface eu posso esses métodos sem saber como eles serão definidos!

Orientação a Objetos



Para que **Jegue** possa ser criada ela precisa obrigatoriamente definir **frear()** e **acelerar()** caso contrário ela terá de ser **abstract** também!

Orientação a Objetos



Para que **Jegue** possa ser criada ela precisa obrigatoriamente definir **frear()** e **acelerar()** caso contrário ela terá de ser **abstract** também!

Orientação a Objetos

- Faça uma classe **Pessoa** que **é um Mamífero**, e implementa as interfaces **Empregado**, **Cidadao**, **Contribuinte**, **Professor**

Orientação a Objetos