DigitalReskilling

Programação Orientada a Objetos

Aula 07 - Overriding

Vitor Santos

Conteúdos

1. Palavras Reservadas
2. Overriding
3. Palavras Reservadas Usadas
4. Links Úteis

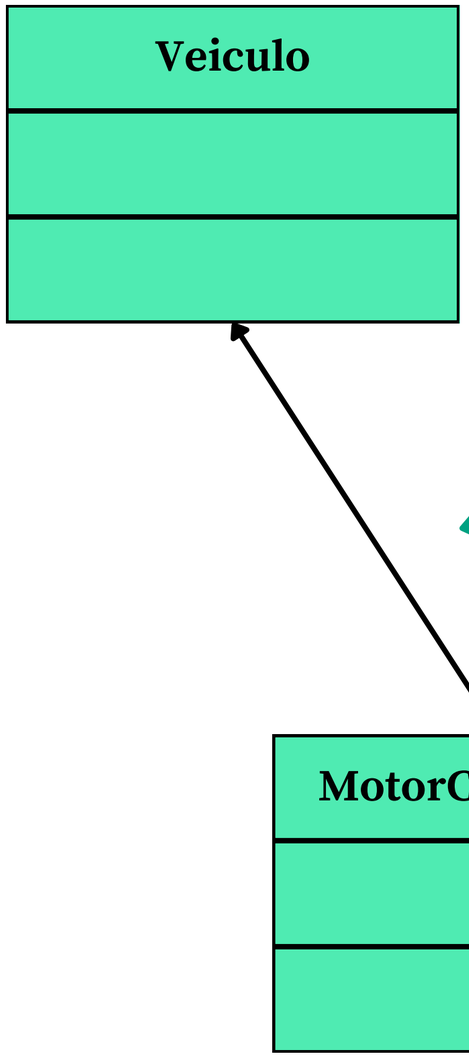
Palavras Reservadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

*** not used**
**** added in 1.2**
***** added in 1.4**
****** added in 5.0**

Conceitos

- Superclass
  - Classe que é uma generalização das suas classes descendentes directas ou não.
- Subclass
  - Classe que é uma especialização daquele que descende.
  - Em Java não há herança múltipla, ou seja, uma class descende de uma única (Object não possui superclass).



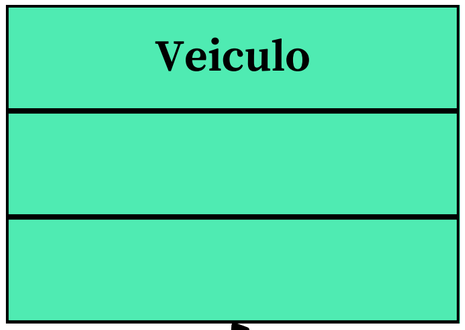
Superclass

Classe Genérica

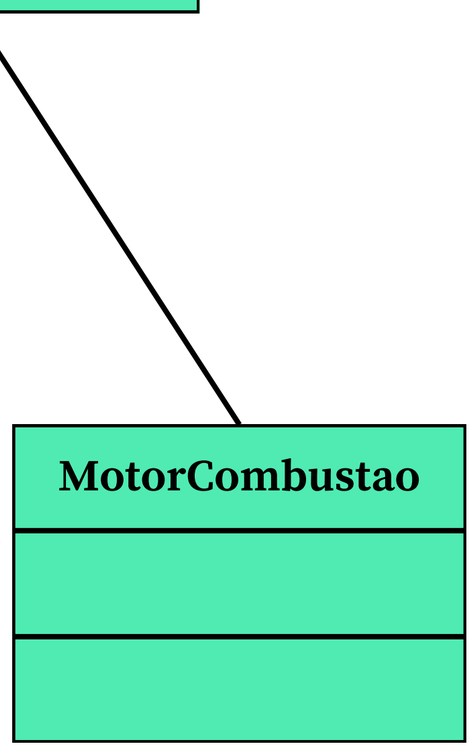
Em Java esta associação é explicitada através da keyword (palavra reservada) **extends**

Subclass

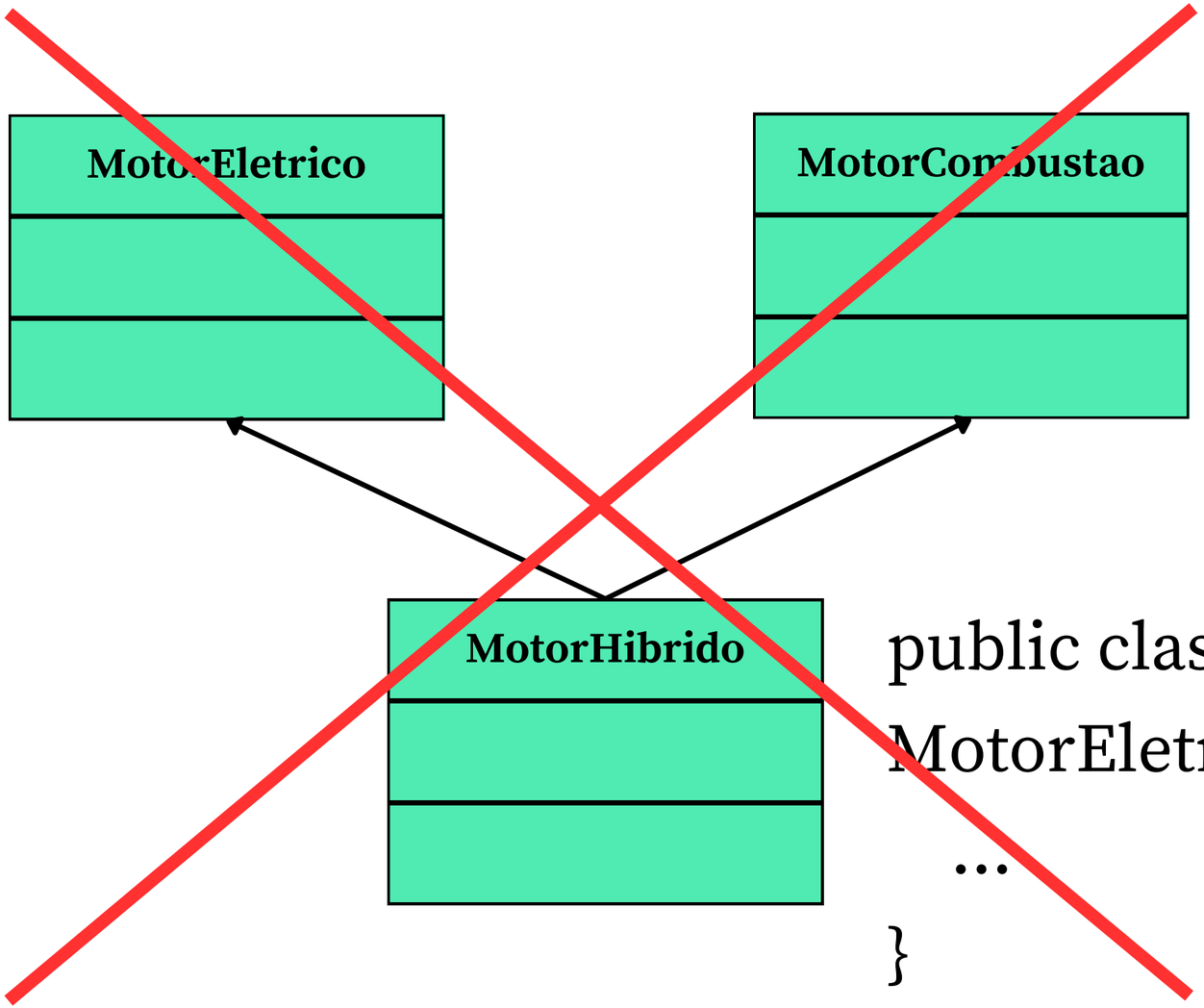
Classe especializada herda todos os membros (atributos, métodos e classes "aninhadas") da superclasse




```
public class Veiculo{
    ...
}
```

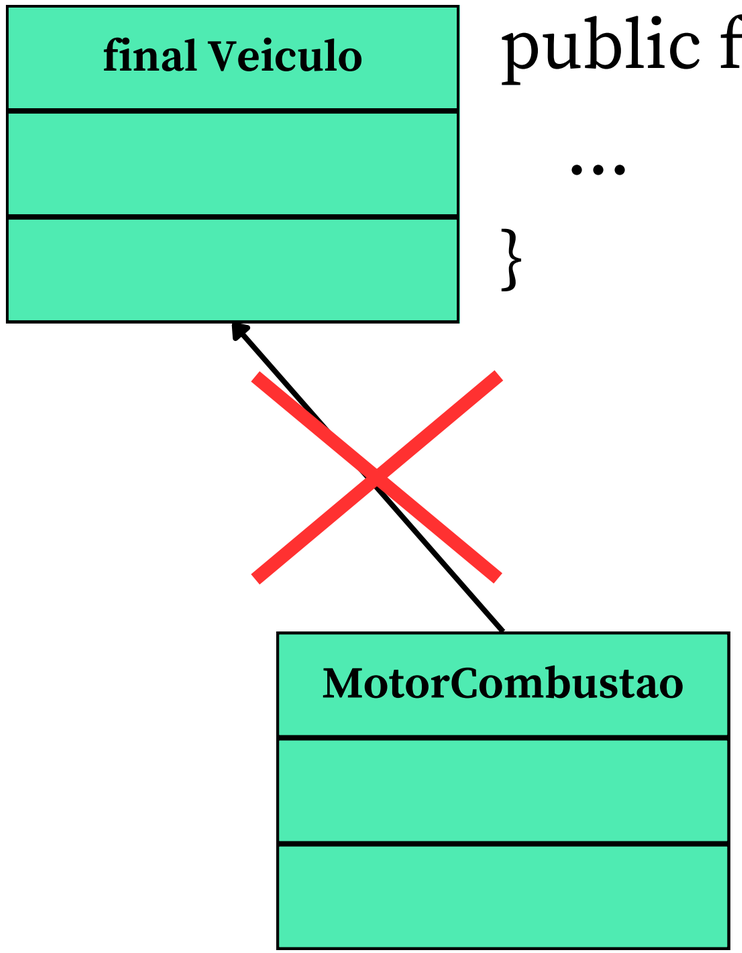



```
public class MotorCombustao extends Veiculo {
    ...
}
```




```
public class MotorHibrido extends
MotorEletrico, MotorCombustao {
    ...
}
```

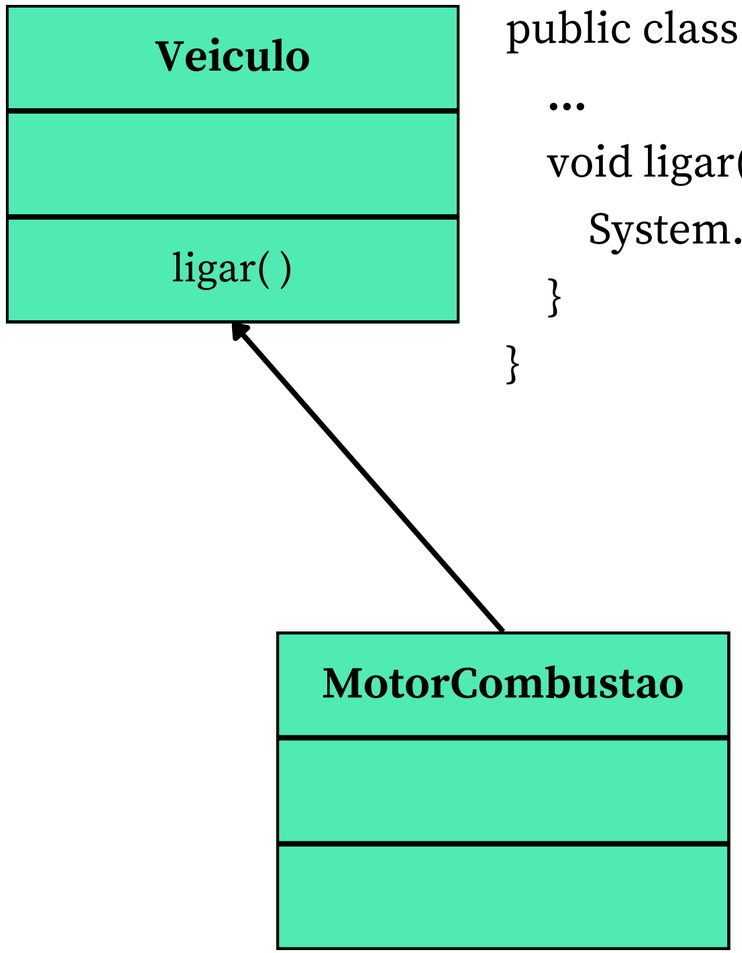
Em Java **não** é possível herança múltipla




```
public final class Veiculo{
    ...
}
```

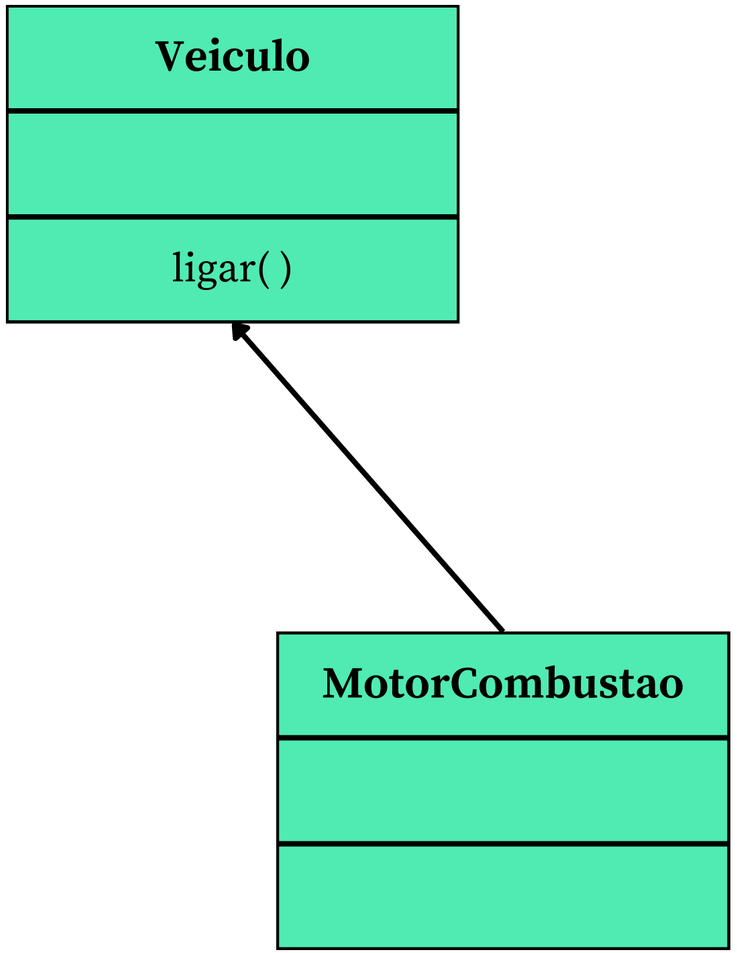
```
public class MotorCombustao extends Veiculo {
    ...
}
```

MotorCombustao **não** pode descender de Veiculo, uma vez que Veiculo foi declarado como **final**

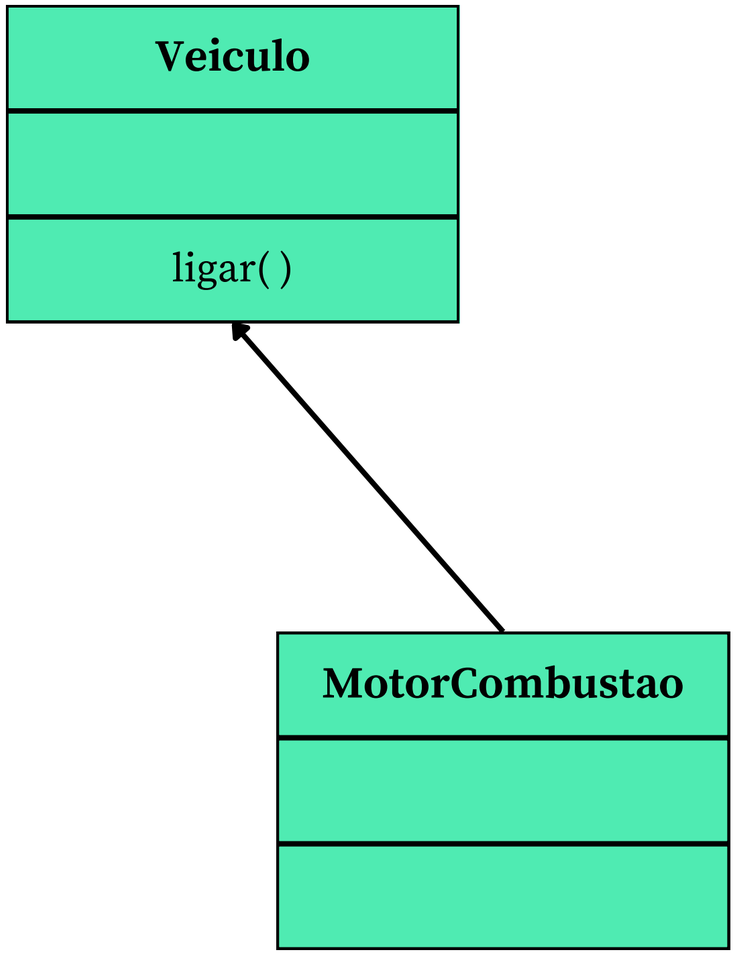



```
public class Veiculo{
   ...
   void ligar(){
      System.out.print("O carro está ligado...");
   }
}
```

MotorCombustao herda o método ligar,
o qual foi assinado na superclass




```
public class Main{
    public static void main(String[] args) {
        MotorCombustao mc = new MotorCombustao( );
        mc.ligar( );
    }
}
```



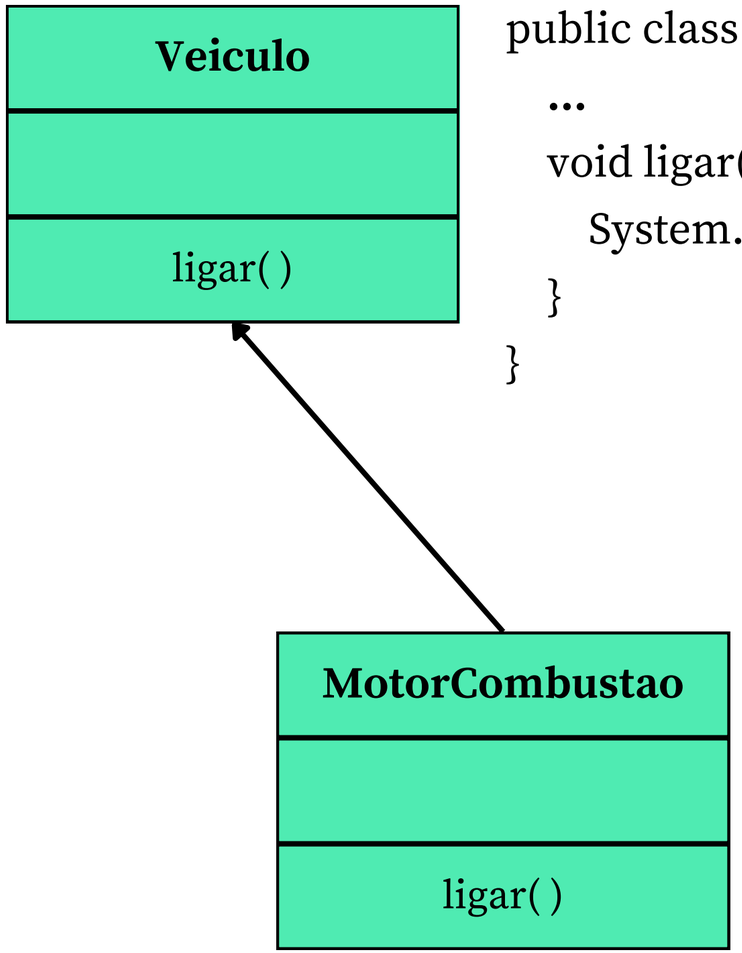

```
public class Main{
    public static void main(String[] args) {
        MotorCombustao mc = new MotorCombustao( );
        mc.ligar( );
    }
}
```

CONSOLA IDE

```
O carro está ligado...
```

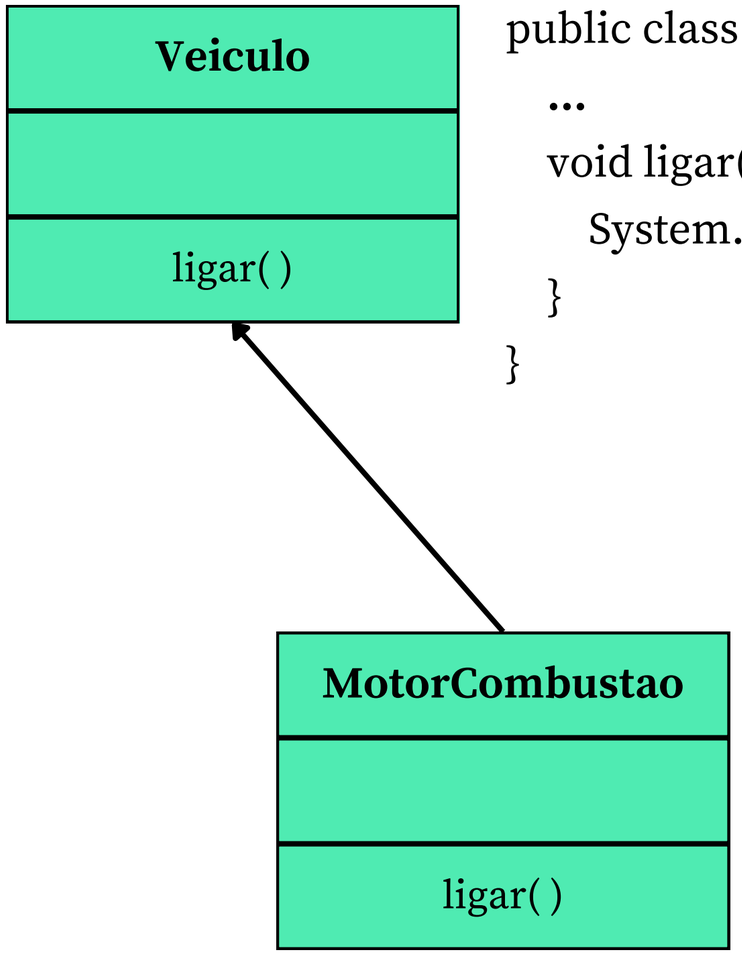
Overriding

- O que fazer se desejarmos que o comportamento de ligar( ) seja outro para MotorCombustao?
- Por exemplo: “O carro está ligado... Vrummm-vrummm!”




```
public class Veiculo{
   ...
   void ligar( ){
      System.out.print("O carro está ligado...");
   }
}
```

```
public class MotorCombustao extends Veiculo {
   ...
   @Override
   void ligar( ){
      System.out.print("O carro está ligado... Vrummm-vrummm!");
   }
}
```




```
public class Veiculo{
   ...
   void ligar( ){
      System.out.print("O carro está ligado...");
   }
}
```

```
public class MotorCombustao extends Veiculo {
   ...
   @Override
   void ligar( ){
      super.ligar( );
      System.out.print("Vrummm-vrummm!");
   }
}
```

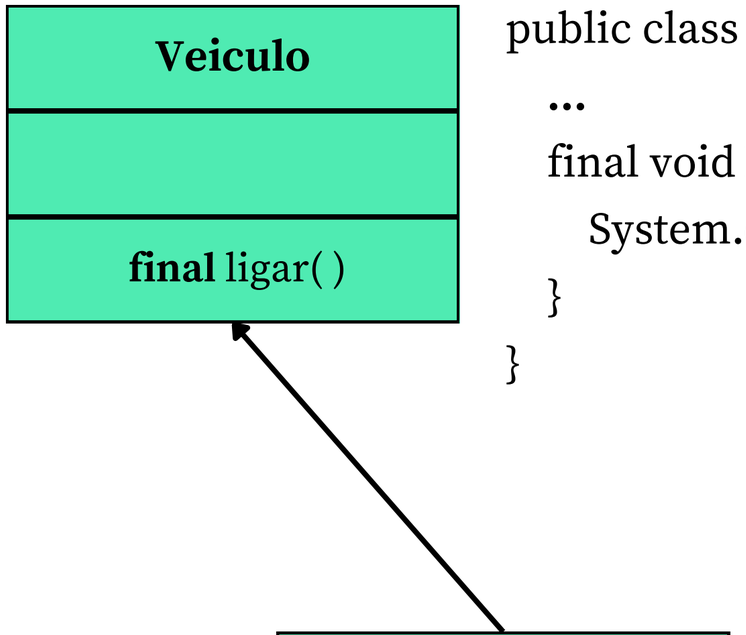
Conceito

- Um método que faça **overriding**, que se sobreponha, a um herdado não pode diminuir a visibilidade em relação ao herdado.

- Se o herdado for **protected** o que sobrepõe **não** pode ser **private**

Conceito

- Não é possível fazer **overriding** de um método que foi assinado como final.




```
public class Veiculo{
   ...
   final void ligar( ){
      System.out.print("O carro está ligado...");
   }
}
```

```
public class MotorCombustao extends Veiculo {
   ...
   @Override
   void ligar( ){
      System.out.print("O carro está ligado... Vrummm-vrummm!");
   }
}
```

Palavras Reservadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

*** not used**
**** added in 1.2**
***** added in 1.4**
****** added in 5.0**