

Programação Orientada a Objetos – Aula 06

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Classes Abstratas

- Classes concretas são utilizadas para gerar os objetos diretamente
- Existem classes que não podem/devem instanciar objetos, são as chamadas **classes abstratas**
- Uma classe abstrata é declarada através da palavra reservada ***abstract***

Classes Abstratas

- Uma classe abstrata serve apenas de modelo para uma classe concreta, por **herança**
- Classes abstratas podem (ou não) possuir **métodos abstratos**, por meio de assinaturas (“sem corpo”)
- Métodos abstratos definidos em uma classe abstrata devem **obrigatoriamente** ser implementados por uma classe concreta

Exemplo de Classe Abstrata

```
Elerodomestico.java ✖  
1 package senac.poo.aula6;  
2  
3 public abstract class Elerodomestico {  
4     private boolean ligado;  
5     private int voltagem;  
6  
7     // métodos abstratos (não possuem corpo)  
8     public abstract void ligar();  
9     public abstract void desligar();  
10  
11     // construtor (não pode instanciar um objeto diretamente)  
12 public Elerodomestico(boolean ligado, int voltagem) {  
13     this.ligado = ligado;  
14     this.voltagem = voltagem;  
15 }  
16
```

Exemplo de Classe Abstrata

```
17 // métodos concretos
18 public void setVoltagem(int voltagem) {
19     this.voltagem = voltagem;
20 }
21
22 public int getVoltagem() {
23     return voltagem;
24 }
25
26 public void setLigado(boolean ligado) {
27     this.ligado = ligado;
28 }
29
30 public boolean isLigado() {
31     return ligado;
32 }
33 }
```

Exemplo de Classe Abstrata

```
TV.java
1 package senac.poo.aula6;
2
3 public class TV extends Eletrodomestico {
4     private int tamanho;
5     private int canal;
6     private int volume;
7
8     public TV(int tamanho, int voltagem) {
9         super(false, voltagem); // construtor da classe abstrata
10        this.tamanho = tamanho;
11        this.canal = 0;
12        this.volume = 0;
13    }
14
15    // implementação dos métodos abstratos
16    public void ligar() {
17        super.setLigado(true);
18        // setCanal(3);
19        // setVolume(25);
20    }
21
22    public void desligar() {
23        super.setLigado(false);
24        // setCanal(0);
25        // setVolume(0);
26    }
27    // falta implementar os métodos setters e getters...
28 }
```

Exemplo de Classe Abstrata

```
Radio.java
1 package senac.poo.aula6;
2
3 public class Radio extends Eletrodomestico {
4
5     public static final short AM = 1;
6     public static final short FM = 2;
7     private int banda;
8     private float sintonia;
9     private int volume;
10
11     public Radio(int voltagem) {
12         super(false, voltagem);
13         // setBanda(Radio.FM);
14         // setSintonia(0);
15         // setVolume(0);
16     }
17
18     // implementação dos métodos abstratos
19     public void ligar() {
20         super.setLigado(true);
21         // setSintonia(88.1f);
22         // setVolume(25);
23     }
24
25     public void desligar() {
26         super.setLigado(false);
27         // setSintonia(0);
28         // setVolume(0);
29     }
30     // falta implementar os métodos setters e getters...
31 }
```

Exemplo de Classe Abstrata

```
Programa.java
1 package senac.poo.aula6;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         TV tv = new TV(29, 110);
8         Radio radio = new Radio(110);
9
10        // chamando métodos abstratos implementados dentro de cada classe (TV e Radio)
11        tv.ligar();
12        radio.ligar();
13
14        System.out.print("Neste momento a TV está " + (tv.isLigado() ? "ligada" : "desligada"));
15        System.out.println(" e o Radio está " + (radio.isLigado() ? "ligado" : "desligado"));
16
17    }
18 }
```

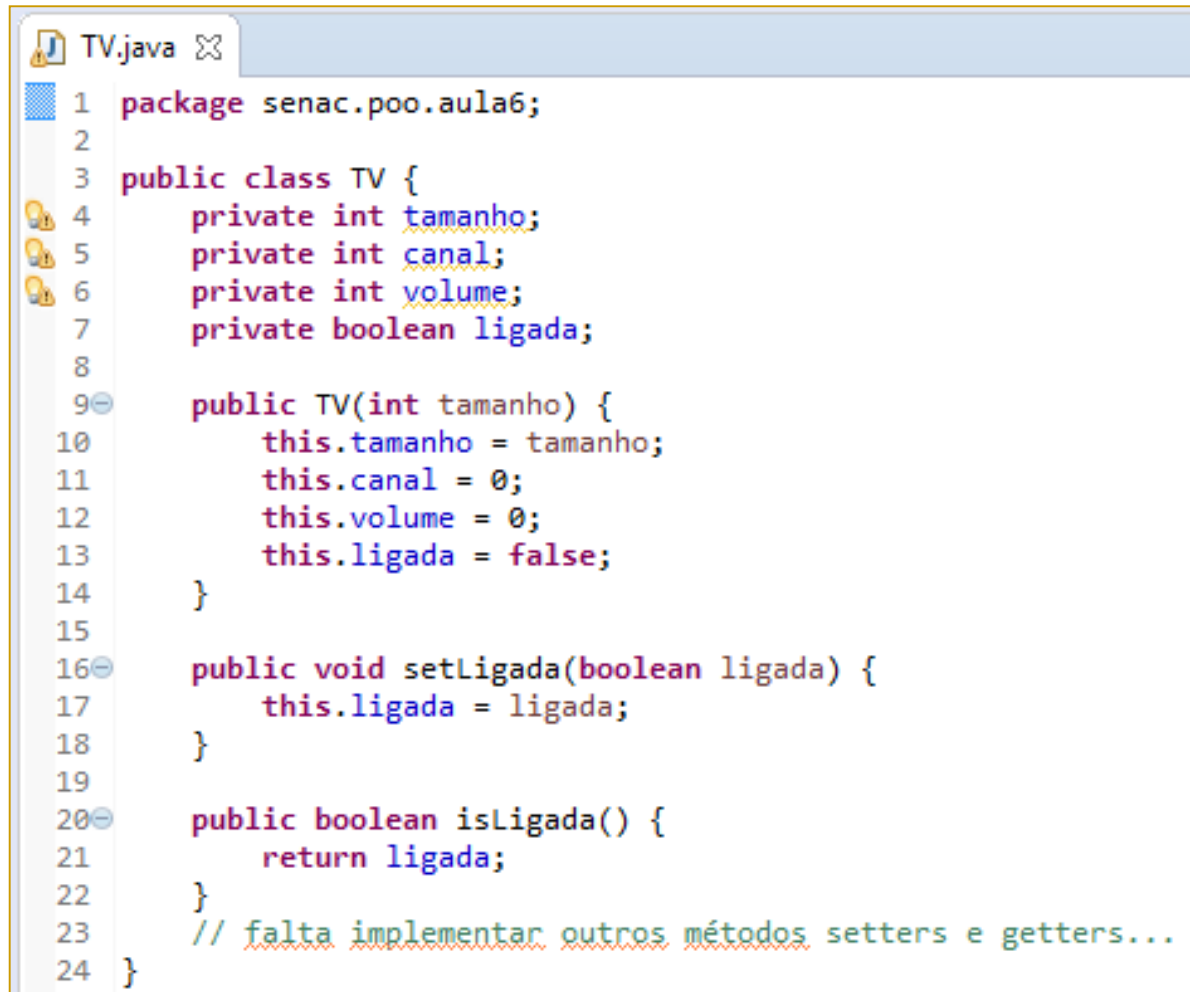

Interfaces

- O uso de herança aumenta o **acoplamento** entre as classes (o quanto uma classe depende de outra)
- Isso acaba fazendo com que o desenvolvedor das subclasses tenha que conhecer a implementação da superclasse e vice-versa
- Isso é uma “dificuldade” de herança mas não de polimorfismo
- Entretanto podemos resolver isso com a ajuda de **interfaces**

Interfaces

- Interface é um “contrato” que define as condições que uma classe deve cumprir para ter um determinado status
- Ao assinar este “contrato” a classe se compromete a explicar como será implementado certos métodos
- Uma interface pode definir uma série de métodos, mas **nunca** conter a implementação deles
- A interface expõe o **comportamento** de um objeto, o que ele **deve** fazer, não **como** ele faz, nem o que ele possui

Exemplo de Interface



```
TV.java
1 package senac.poo.aula6;
2
3 public class TV {
4     private int tamanho;
5     private int canal;
6     private int volume;
7     private boolean ligada;
8
9     public TV(int tamanho) {
10         this.tamanho = tamanho;
11         this.canal = 0;
12         this.volume = 0;
13         this.ligada = false;
14     }
15
16     public void setLigada(boolean ligada) {
17         this.ligada = ligada;
18     }
19
20     public boolean isLigada() {
21         return ligada;
22     }
23     // falta implementar outros métodos setters e getters...
24 }
```

Exemplo de Interface

ControleRemoto.java

```
1 package senac.poo.aula6;
2
3 public interface ControleRemoto {
4
5     /*
6      * Perceba que temos apenas as assinaturas dos métodos,
7      * e que cada método termina com um ponto-e-vírgula (;)
8      */
9
10    void ligar();
11    void desligar();
12    void mudarCanal(int Canal);
13    void aumentarVolume(int taxa);
14    void diminuirVolume(int taxa);
15 }
```

Exemplo de Interface

```
ModeloTV001.java
1 package senac.poo.aula6;
2
3 public class ModeloTV001 extends TV implements ControleRemoto {
4     public String MODELO = "TV-001";
5
6     public ModeloTV001(int tamanho) {
7         super(tamanho);
8     }
9
10    // implementação dos métodos da interface ControleRemoto
11    public void ligar() {
12        super.setLigada(true);
13    }
14
15    public void desligar() {
16        super.setLigada(false);
17    }
18
19    public void mudarCanal(int Canal) { }
20    public void aumentarVolume(int taxa) { }
21    public void diminuirVolume(int taxa) { }
22
23 }
```

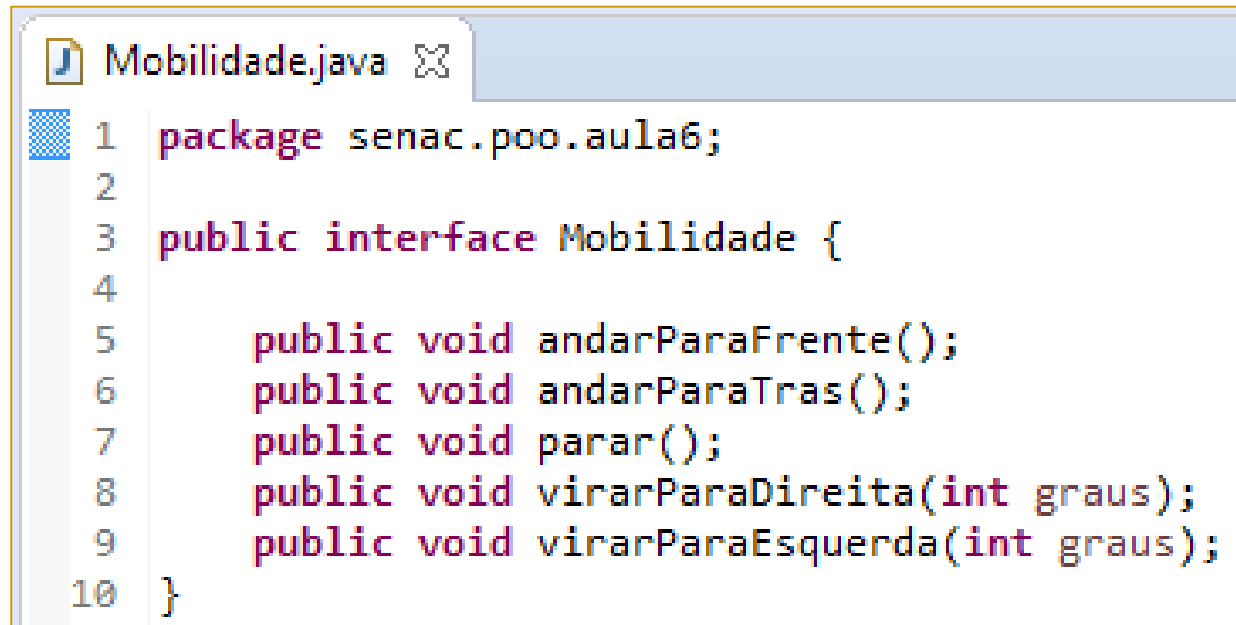
Exemplo de Interface

```
ModeloSDX.java ✕
1 package senac.poo.aula6;
2
3 public class ModeloSDX extends TV implements ControleRemoto{
4     public String MODELO = "TV-SDX";
5
6     public ModeloSDX(int tamanho) {
7         super(tamanho);
8     }
9
10    // implementação dos métodos da interface ControleRemoto
11    public void ligar() {
12        super.setLigada(true);
13    }
14
15    public void desligar() {
16        System.out.println("Hasta la vista, baby!");
17        super.setLigada(false);
18    }
19
20    public void mudarCanal(int Canal) { }
21    public void aumentarVolume(int taxa) { }
22    public void diminuirVolume(int taxa) { }
23 }
```

Exemplo de Interface

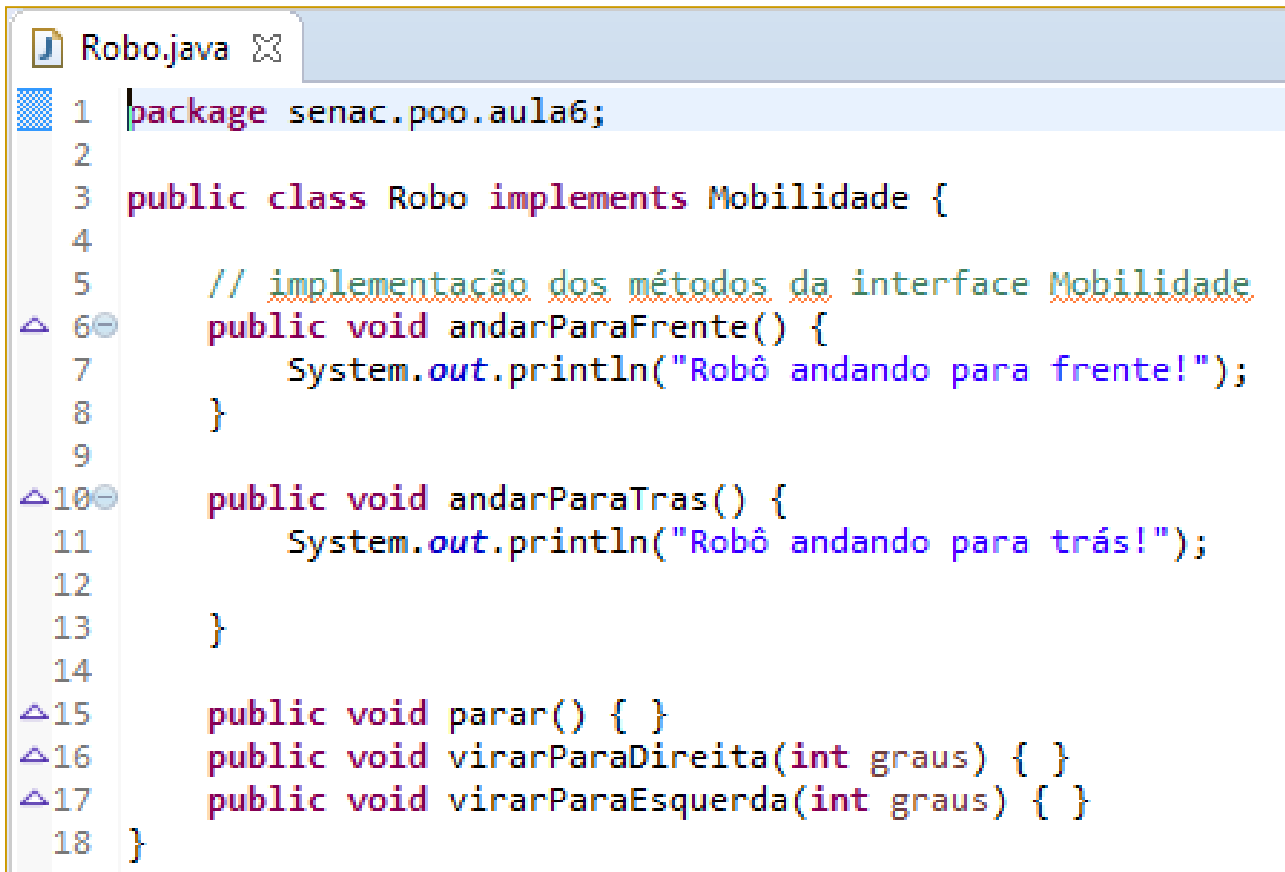
```
Programa.java ✕
1 package senac.poo.aula6;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         ModeloTV001 tv1 = new ModeloTV001(21);
8         ModeloSDX tv2 = new ModeloSDX(42);
9
10        tv1.ligar();
11        tv2.ligar();
12
13        System.out.println("TV modelo " + tv1.MODELO + " está " + (tv1.isLigada() ? "ligada" : "desligada"));
14        System.out.println("TV modelo " + tv2.MODELO + " está " + (tv2.isLigada() ? "ligada" : "desligada"));
15
16        System.out.println("Desligando a TV modelo " + tv1.MODELO);
17        tv1.desligar();
18        System.out.println("Desligando a TV modelo " + tv2.MODELO);
19        tv2.desligar();
20    }
21 }
```

Outro Exemplo de Interface



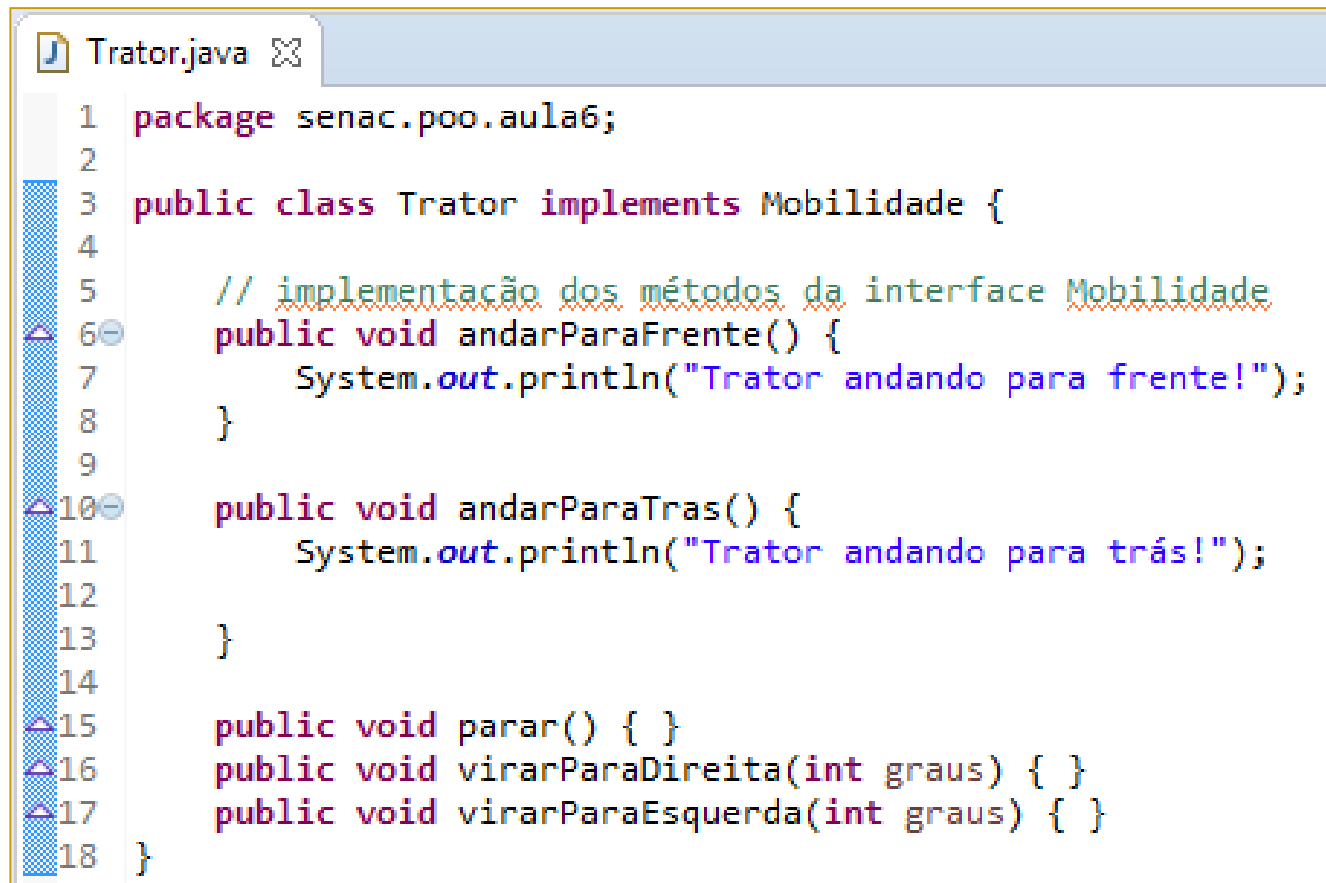
```
Mobilidade.java ✖
1 package senac.poo.aula6;
2
3 public interface Mobilidade {
4
5     public void andarParaFrente();
6     public void andarParaTras();
7     public void parar();
8     public void virarParaDireita(int graus);
9     public void virarParaEsquerda(int graus);
10 }
```


Outro Exemplo de Interface



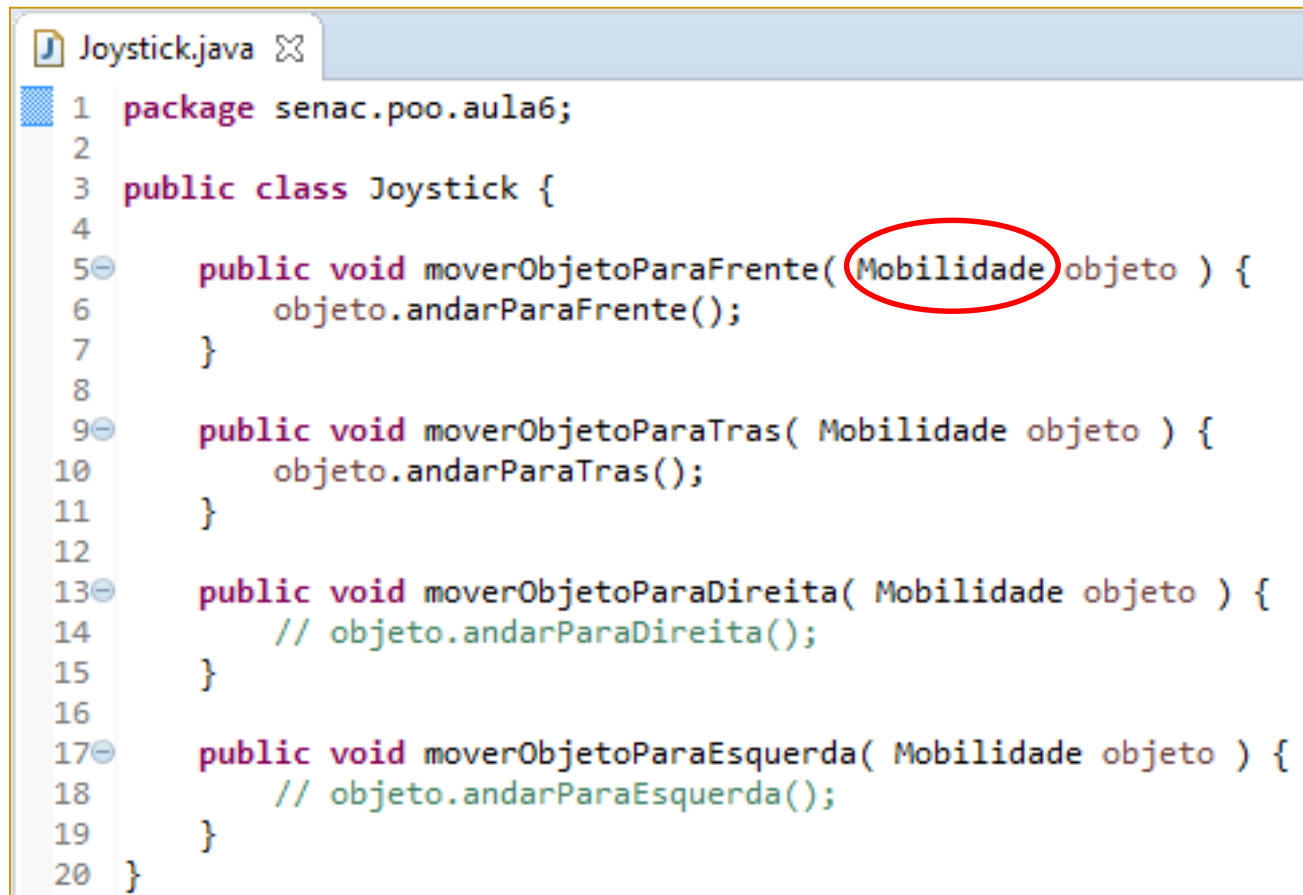
```
Robo.java ✕
1 package senac.poo.aula6;
2
3 public class Robo implements Mobilidade {
4
5     // implementação dos métodos da interface Mobilidade
6     public void andarParaFrente() {
7         System.out.println("Robô andando para frente!");
8     }
9
10    public void andarParaTras() {
11        System.out.println("Robô andando para trás!");
12    }
13
14
15    public void parar() { }
16    public void virarParaDireita(int graus) { }
17    public void virarParaEsquerda(int graus) { }
18 }
```

Outro Exemplo de Interface



```
Trator.java ✕
1 package senac.poo.aula6;
2
3 public class Trator implements Mobilidade {
4
5     // implementação dos métodos da interface Mobilidade
6     public void andarParaFrente() {
7         System.out.println("Trator andando para frente!");
8     }
9
10    public void andarParaTras() {
11        System.out.println("Trator andando para trás!");
12    }
13
14
15    public void parar() { }
16    public void virarParaDireita(int graus) { }
17    public void virarParaEsquerda(int graus) { }
18 }
```

Outro Exemplo de Interface



```
Joystick.java
1 package senac.poo.aula6;
2
3 public class Joystick {
4
5     public void moverObjetoParaFrente( Mobilidade objeto ) {
6         objeto.andarParaFrente();
7     }
8
9     public void moverObjetoParaTras( Mobilidade objeto ) {
10        objeto.andarParaTras();
11    }
12
13    public void moverObjetoParaDireita( Mobilidade objeto ) {
14        // objeto.andarParaDireita();
15    }
16
17    public void moverObjetoParaEsquerda( Mobilidade objeto ) {
18        // objeto.andarParaEsquerda();
19    }
20 }
```

Outro Exemplo de Interface

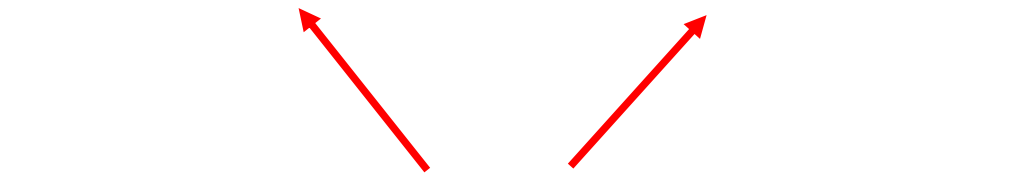
```
Programa.java ✖
1 package senac.poo.aula6;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         Robo robo = new Robo();
8         Trator trator = new Trator();
9         Joystick joystick = new Joystick();
10
11         joystick.moverObjetoParaFrente(robo);
12
13         joystick.moverObjetoParaTras(trator);
14
15     }
16 }
```

“Emulando” herança múltipla no Java

```
Contrato1.java ✕  
1 package senac.poo.aula6;  
2  
3 public interface Contrato1 {  
4  
5     public void metodoContrato1();  
6 }
```

```
Contrato2.java ✕  
1 package senac.poo.aula6;  
2  
3 public interface Contrato2 {  
4  
5     public void metodoContrato2();  
6 }
```

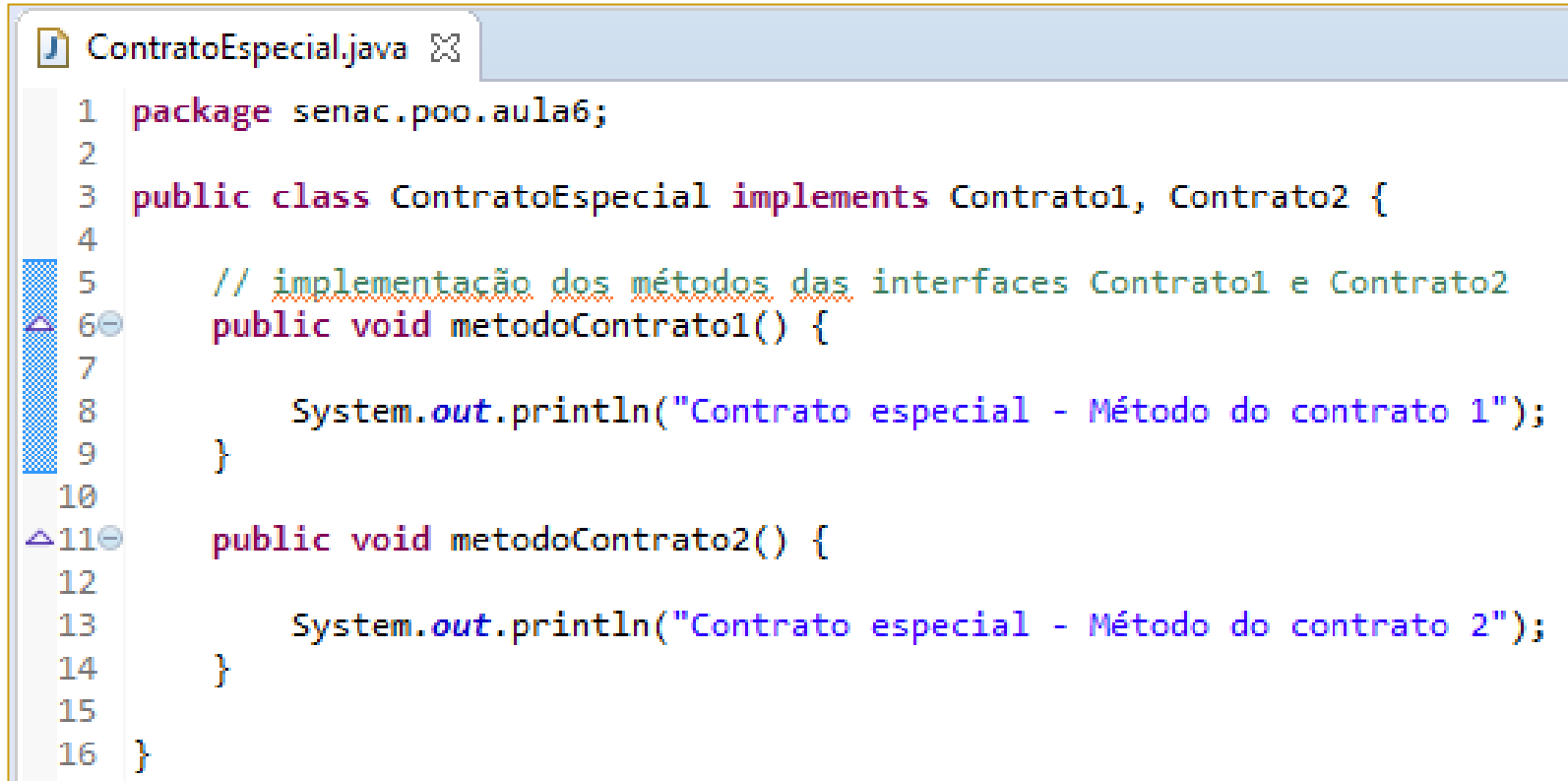
```
Contrato3.java ✕  
1 package senac.poo.aula6;  
2  
3 public interface Contrato3 extends Contrato1, Contrato2 {  
4  
5     public void metodoContrato3();  
6 }
```



“Emulando” herança múltipla no Java

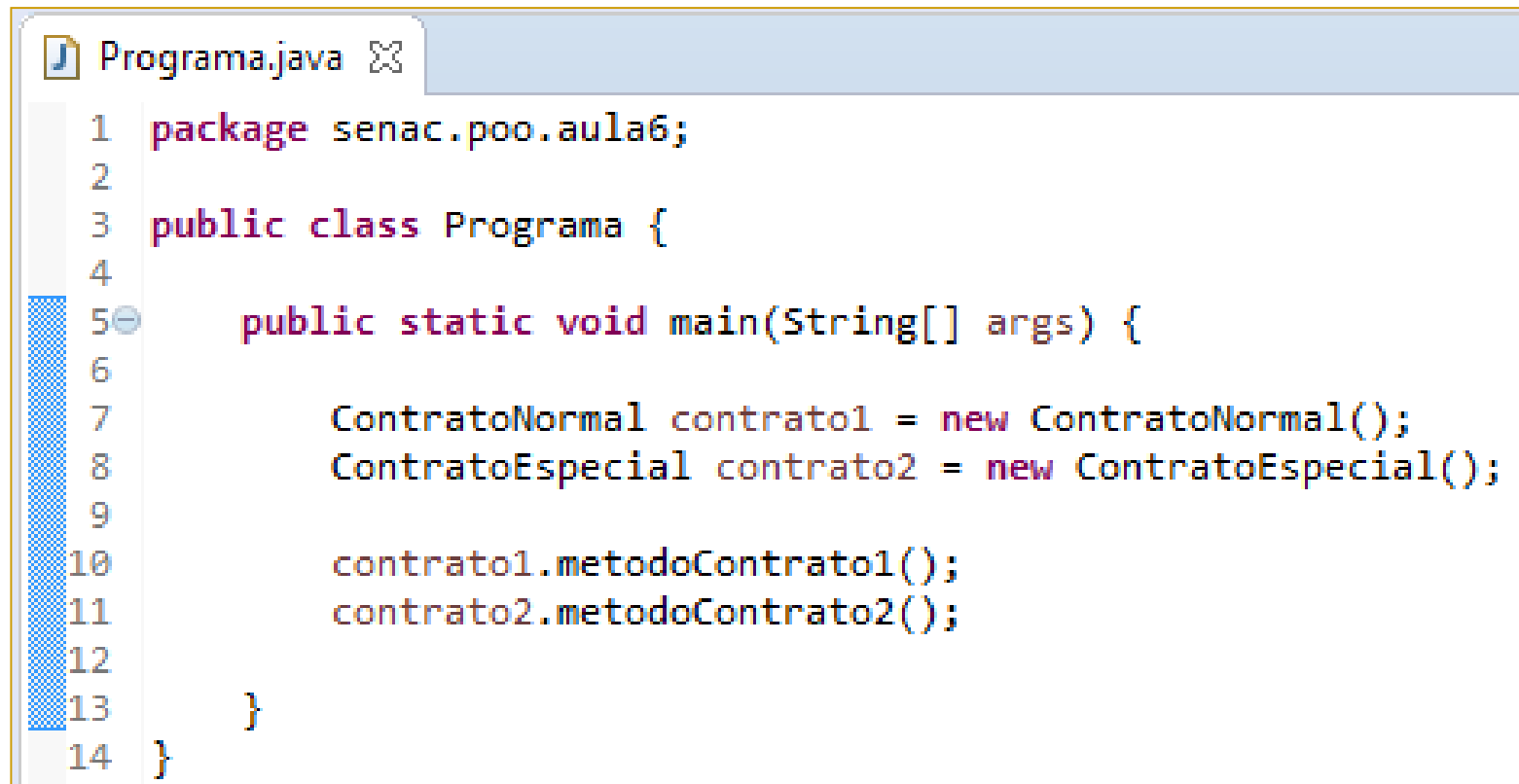
```
ContratoNormal.java ✕
1 package senac.poo.aula6;
2
3 public class ContratoNormal implements Contrato3 {
4
5     // implementação dos métodos da interface Contrato3 (inclui Contrato1 e Contrato2)
6     public void metodoContrato1() {
7
8         System.out.println("Contrato normal - Método do contrato 1");
9     }
10
11     public void metodoContrato2() {
12
13         System.out.println("Contrato normal - Método do contrato 2");
14     }
15
16     public void metodoContrato3() {
17
18         System.out.println("Contrato normal - Método do contrato 2");
19     }
20 }
```

“Emulando” herança múltipla no Java



```
1 package senac.poo.aula6;
2
3 public class ContratoEspecial implements Contrato1, Contrato2 {
4
5     // implementação dos métodos das interfaces Contrato1 e Contrato2
6     public void metodoContrato1() {
7
8         System.out.println("Contrato especial - Método do contrato 1");
9     }
10
11     public void metodoContrato2() {
12
13         System.out.println("Contrato especial - Método do contrato 2");
14     }
15
16 }
```

“Emulando” herança múltipla no Java



```
Programa.java ✕
1 package senac.poo.aula6;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         ContratoNormal contrato1 = new ContratoNormal();
8         ContratoEspecial contrato2 = new ContratoEspecial();
9
10        contrato1.metodoContrato1();
11        contrato2.metodoContrato2();
12
13    }
14 }
```