

Aula 19 – Especificadores de Acesso e Pacotes

Norton Trevisan Roman

23 de maio de 2013

Private

```
class Casa {  
    private double valorM2 = 1500;  
  
    Casa() {}  
  
    Casa(double valorM2) {  
        this.valorM2 = valorM2;  
    }  
  
    ...  
}
```

- O que acontece quando compilamos *CasaQuad*?

```
class CasaQuad extends Casa {  
    double lateral = 10;  
  
    CasaQuad() {}  
  
    CasaQuad(double valorM2) {  
        super(valorM2);  
    }  
  
    CasaQuad(double lateral, double valorM2) {  
        super(valorM2);  
        this.lateral = lateral;  
    }  
  
    double area() {  
        double areat=-1;  
  
        if (this.lateral>=0) {  
            areat = this.lateral*this.lateral;  
        }  
        return(areat);  
    }  
}
```

Private

```
class Casa {  
    private double valorM2 = 1500;  
  
    Casa() {}  
  
    Casa(double valorM2) {  
        this.valorM2 = valorM2;  
    }  
  
    ...  
}
```

- O que acontece quando compilamos *CasaQuad*?

Linha de Comando

```
$ javac CasaQuad.java
```

```
class CasaQuad extends Casa {  
    double lateral = 10;  
  
    CasaQuad() {}  
  
    CasaQuad(double valorM2) {  
        super(valorM2);  
    }  
  
    CasaQuad(double lateral, double valorM2) {  
        super(valorM2);  
        this.lateral = lateral;  
    }  
  
    double area() {  
        double areat=-1;  
  
        if (this.lateral>=0) {  
            areat = this.lateral*this.lateral;  
        }  
        return(areat);  
    }  
}
```

Private

```
class Casa {  
    private double valorM2 = 1500;  
  
    Casa() {}  
  
    Casa(double valorM2) {  
        this.valorM2 = valorM2;  
    }  
  
    ...  
}
```

- O que acontece quando compilamos *CasaQuad*?

Linha de Comando

```
$ javac CasaQuad.java
```

- ▶ Compila. Nada impede.

```
class CasaQuad extends Casa {  
    double lateral = 10;  
  
    CasaQuad() {}  
  
    CasaQuad(double valorM2) {  
        super(valorM2);  
    }  
  
    CasaQuad(double lateral, double valorM2) {  
        super(valorM2);  
        this.lateral = lateral;  
    }  
  
    double area() {  
        double areat=-1;  
  
        if (this.lateral>=0) {  
            areat = this.lateral*this.lateral;  
        }  
        return(areat);  
    }  
}
```

- E quando compilamos *CasaRet*?

```
class CasaRet extends Casa {
    double cquarto = 10;
    double lateral = 10;

    CasaRet() {}

    CasaRet(double valorM2) {
        super(valorM2);
    }

    CasaRet(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    CasaRet(double lateral, double cquarto,
            double valorM2) {
        this(lateral, cquarto);
        this.valorM2 = valorM2;
    }

    double area() {
        double areat=-1;

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }
}
```

- E quando compilamos *CasaRet*?

Linha de Comando

```
$ javac CasaRet.java
CasaRet.java:35: valorM2 has private access
in Casa
    this.valorM2 = valorM2;
    ~
1 error
```

```
class CasaRet extends Casa {
    double cquarto = 10;
    double lateral = 10;

    CasaRet() {}

    CasaRet(double valorM2) {
        super(valorM2);
    }

    CasaRet(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    CasaRet(double lateral, double cquarto,
            double valorM2) {
        this(lateral, cquarto);
        this.valorM2 = valorM2;
    }

    double area() {
        double areat=-1;

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }
}
```

- E quando compilamos *CasaRet*?

Linha de Comando

```
$ javac CasaRet.java
CasaRet.java:35: valorM2 has private access
in Casa
    this.valorM2 = valorM2;
    ^
1 error
```

- ▶ Não temos acesso ao valor do m^2

```
class CasaRet extends Casa {
    double cquarto = 10;
    double lateral = 10;

    CasaRet() {}

    CasaRet(double valorM2) {
        super(valorM2);
    }

    CasaRet(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    CasaRet(double lateral, double cquarto,
            double valorM2) {
        this(lateral, cquarto);
        this.valorM2 = valorM2;
    }

    double area() {
        double areat=-1;

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }
}
```

- E quando compilamos *CasaRet*?

Linha de Comando

```
$ javac CasaRet.java
CasaRet.java:35: valorM2 has private access
in Casa
    this.valorM2 = valorM2;
    ~
1 error
```

- ▶ Não temos acesso ao valor do m^2
- ▶ Muito embora ele esteja na memória do objeto!

```
class CasaRet extends Casa {
    double cquarto = 10;
    double lateral = 10;

    CasaRet() {}

    CasaRet(double valorM2) {
        super(valorM2);
    }

    CasaRet(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    CasaRet(double lateral, double cquarto,
            double valorM2) {
        this(lateral, cquarto);
        this.valorM2 = valorM2;
    }

    double area() {
        double areat=-1;

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }
}
```


- E quando compilamos *CasaRet*?

Linha de Comando

```
$ javac CasaRet.java
CasaRet.java:35: valorM2 has private access
in Casa
    this.valorM2 = valorM2;
    ~
1 error
```

- ▶ Não temos acesso ao valor do m^2
 - ▶ Muito embora ele esteja na memória do objeto!
- Que fazer?

```
class CasaRet extends Casa {
    double cquarto = 10;
    double lateral = 10;

    CasaRet() {}

    CasaRet(double valorM2) {
        super(valorM2);
    }

    CasaRet(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    CasaRet(double lateral, double cquarto,
            double valorM2) {
        this(lateral, cquarto);
        this.valorM2 = valorM2;
    }

    double area() {
        double areat=-1;

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }
}
```

Private

- E quando compilamos *CasaRet*?

Linha de Comando

```
$ javac CasaRet.java
CasaRet.java:35: valorM2 has private access
in Casa
    this.valorM2 = valorM2;
    ~
1 error
```

- ▶ Não temos acesso ao valor do m^2
- ▶ Muito embora ele esteja na memória do objeto!
- Que fazer?
 - ▶ Refatorar a classe

```
class CasaRet extends Casa {
    double cquarto = 10;
    double lateral = 10;

    CasaRet() {}

    CasaRet(double valorM2) {
        super(valorM2);
    }

    CasaRet(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    CasaRet(double lateral, double cquarto,
            double valorM2) {
        super(valorM2);
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    double area() {
        double areat=-1;

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }
}
```

Herança

- E não poderíamos ter *this* e *super* juntas?

Herança

- E não poderíamos ter *this* e *super* juntas?

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        super(valorM2);  
        this(lateral,cquarto);  
    }  
}
```

Herança

- E não poderíamos ter *this* e *super* juntas?

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        super(valorM2);  
        this(lateral,cquarto);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to this must be first  
statement in constructor  
        this(lateral,cquarto);  
        ^  
1 error
```

Herança

- E não poderíamos ter *this* e *super* juntas?

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        super(valorM2);  
        this(lateral,cquarto);  
    }  
}
```

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        this(lateral,cquarto);  
        super(valorM2);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to this must be first  
statement in constructor  
        this(lateral,cquarto);  
        ^  
1 error
```

Herança

- E não poderíamos ter *this* e *super* juntas?

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        super(valorM2);  
        this(lateral,cquarto);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to this must be first  
statement in constructor  
        this(lateral,cquarto);  
        ^  
1 error
```

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        this(lateral,cquarto);  
        super(valorM2);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to super must be first  
statement in constructor  
        super(valorM2);  
        ^  
1 error
```

Herança

- E não poderíamos ter *this* e *super* juntas?

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        super(valorM2);  
        this(lateral,cquarto);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to this must be first  
statement in constructor  
    this(lateral,cquarto);  
    ^  
1 error
```

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        this(lateral,cquarto);  
        super(valorM2);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to super must be first  
statement in constructor  
    super(valorM2);  
    ^  
1 error
```

- Como, se ambas devem ser o primeiro comando?

Herança

- E não poderíamos ter *this* e *super* juntas?

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        super(valorM2);  
        this(lateral,cquarto);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to this must be first  
statement in constructor  
    this(lateral,cquarto);  
    ^  
1 error
```

```
class CasaRet extends Casa {  
    ...  
    CasaRet(double lateral, double cquarto,  
            double valorM2) {  
        this(lateral,cquarto);  
        super(valorM2);  
    }  
}
```

Linha de Comando

```
$ javac CasaRet.java  
CasaRet.java:35: call to super must be first  
statement in constructor  
    super(valorM2);  
    ^  
1 error
```

- Como, se ambas devem ser o primeiro comando?
 - ▶ Teremos que optar

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso
 - ▶ Public:

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso
 - ▶ Public:
 - ★ *public classe, construtor, atributo ou método*: Acesso irrestrito

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso
 - ▶ Public:
 - ★ *public classe, construtor, atributo ou método*: Acesso irrestrito
 - ▶ Protected:

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso
 - ▶ Public:
 - ★ *public classe, construtor, atributo ou método*: Acesso irrestrito
 - ▶ Protected:
 - ★ *protected construtor, atributo ou método*: Somente elementos pertencentes a classes do mesmo pacote, ou subclasse (independente do pacote) têm acesso

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso
 - ▶ Public:
 - ★ *public classe, construtor, atributo ou método*: Acesso irrestrito
 - ▶ Protected:
 - ★ *protected construtor, atributo ou método*: Somente elementos pertencentes a classes do mesmo pacote, ou subclasse (independente do pacote) têm acesso
 - ▶ Nada (“friendly”)

Especificadores (ou Modificadores) de Acesso

- Determinam quem terá permissão para acessar ou modificar os componentes (atributos e métodos) das classes.
 - ▶ Private:
 - ★ *private construtor, atributo ou método*: Somente elementos de dentro da classe têm acesso
 - ▶ Public:
 - ★ *public classe, construtor, atributo ou método*: Acesso irrestrito
 - ▶ Protected:
 - ★ *protected construtor, atributo ou método*: Somente elementos pertencentes a classes do mesmo pacote, ou subclasse (independente do pacote) têm acesso
 - ▶ Nada (“friendly”)
 - ★ *classe, construtor, atributo ou método*: Somente elementos pertencentes a classes do mesmo pacote têm acesso

Pacotes

- Pacote?

Pacotes

- Pacote?
 - ▶ Corresponde ao modo como organizamos os arquivos .java em diferentes diretórios

Pacotes

- Pacote?
 - ▶ Corresponde ao modo como organizamos os arquivos .java em diferentes diretórios
 - ▶ É o diretório

Pacotes

- Pacote?
 - ▶ Corresponde ao modo como organizamos os arquivos .java em diferentes diretórios
 - ▶ É o diretório
- Podem ser declarados explicitamente, com a palavra *package*

Pacotes

- Pacote?
 - ▶ Corresponde ao modo como organizamos os arquivos .java em diferentes diretórios
 - ▶ É o diretório
- Podem ser declarados explicitamente, com a palavra *package*
 - ▶ Útil quando temos duas classes com o mesmo nome, mas funções diferentes.

Pacotes

- Pacote?
 - ▶ Corresponde ao modo como organizamos os arquivos .java em diferentes diretórios
 - ▶ É o diretório
- Podem ser declarados explicitamente, com a palavra *package*
 - ▶ Útil quando temos duas classes com o mesmo nome, mas funções diferentes.
 - ★ Ex: *normal.Matematica* e *complexo.Matematica*, que tratariam de funções matemáticas para números complexos e os demais

- Pacote?
 - ▶ Corresponde ao modo como organizamos os arquivos .java em diferentes diretórios
 - ▶ É o diretório
- Podem ser declarados explicitamente, com a palavra *package*
 - ▶ Útil quando temos duas classes com o mesmo nome, mas funções diferentes.
 - ★ Ex: *normal.Matematica* e *complexo.Matematica*, que tratariam de funções matemáticas para números complexos e os demais
 - ★ nesse caso, a primeira seria *normal/Matematica.class* e a segunda em *complexos/Matematica.class*

Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?

Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina

Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes

Pacotes

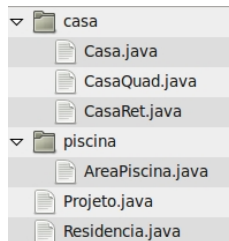
- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa*

Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa* e *piscina*

Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa e piscina*



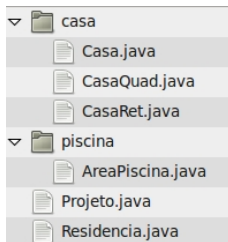
Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa e piscina*

- E...

Casa.java:

```
package casa;  
class Casa { ... }
```



Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa e piscina*

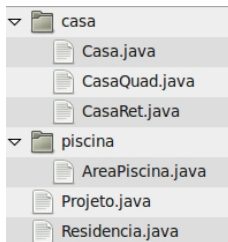
- E...

Casa.java:

```
package casa;  
class Casa { ... }
```

CasaQuad.java:

```
package casa;  
class CasaQuad extends Casa { .. }
```



Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa e piscina*

- E...

Casa.java:

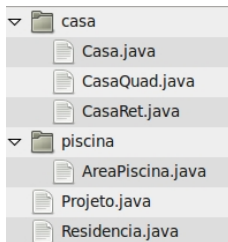
```
package casa;  
class Casa { ... }
```

CasaQuad.java:

```
package casa;  
class CasaQuad extends Casa { .. }
```

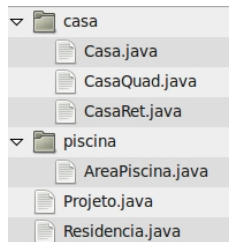
CasaRet.java:

```
package casa;  
class CasaRet extends Casa { ... }
```



Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
 - ▶ Poderíamos separar coisas relativas a casa, das relativas a piscina
 - ▶ 2 pacotes
 - ★ *casa e piscina*



- E...

Casa.java:

```
package casa;  
class Casa { ... }
```

CasaQuad.java:

```
package casa;  
class CasaQuad extends Casa { .. }
```

CasaRet.java:

```
package casa;  
class CasaRet extends Casa { ... }
```

AreaPiscina.java:

```
package piscina;  
class AreaPiscina { ... }
```

Pacotes

- E como compilamos isso?

Pacotes

- E como compilamos isso?
 - ▶ Do diretório onde estão *casa* e *piscina*

Pacotes

- E como compilamos isso?
 - ▶ Do diretório onde estão *casa* e *piscina*

```
$ javac casa/Casa.java  
$ javac casa/CasaRet.java  
$ javac casa/CasaQuad.java  
$ javac piscina/AreaPiscina.java
```

Pacotes

- E como compilamos isso?
 - ▶ Do diretório onde estão *casa* e *piscina*

```
$ javac casa/Casa.java  
$ javac casa/CasaRet.java  
$ javac casa/CasaQuad.java  
$ javac piscina/AreaPiscina.java
```

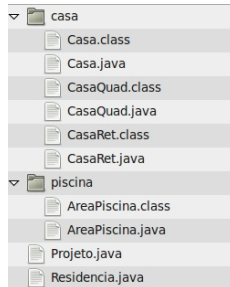
- E o diretório conterá

Pacotes

- E como compilamos isso?
 - ▶ Do diretório onde estão *casa* e *piscina*

```
$ javac casa/Casa.java  
$ javac casa/CasaRet.java  
$ javac casa/CasaQuad.java  
$ javac piscina/AreaPiscina.java
```

- E o diretório conterá

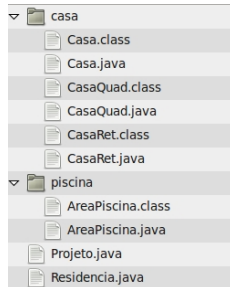


Pacotes

- E como compilamos isso?
 - ▶ Do diretório onde estão *casa* e *piscina*

```
$ javac casa/Casa.java  
$ javac casa/CasaRet.java  
$ javac casa/CasaQuad.java  
$ javac piscina/AreaPiscina.java
```

- E o diretório conterá
 - ▶ Note os .class



- E o que acontece ao compilarmos *Residencia*?

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

Pacotes

- E o que acontece ao compilarmos *Residencia*?

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

Linha de Comando

```
$ javac Residencia.java  
Residencia.java:6: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Casa casa;  
    ^  
Residencia.java:8: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    AreaPiscina piscina;  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
    piscina) {  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
    piscina) {  
    ^  
4 errors
```

Pacotes

- E o que acontece ao compilarmos *Residencia*?

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

- ▶ O compilador não sabe onde está nem *Casa* nem *AreaPiscina*

Linha de Comando

```
$ javac Residencia.java  
Residencia.java:6: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Casa casa;  
    ^  
Residencia.java:8: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    AreaPiscina piscina;  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
                piscina) {  
                ^  
Residencia.java:13: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
                piscina) {  
                ^  
4 errors
```

Pacotes

- E o que acontece ao compilarmos *Residencia*?

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

- ▶ O compilador não sabe onde está nem *Casa* nem *AreaPiscina*
- ▶ Irá buscar sempre no mesmo diretório do arquivo compilado.

Linha de Comando

```
$ javac Residencia.java  
Residencia.java:6: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Casa casa;  
    ^  
Residencia.java:8: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    AreaPiscina piscina;  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
    piscina) {  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
    piscina) {  
    ^  
4 errors
```

Pacotes

- E o que acontece ao compilarmos *Residencia*?

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

- ▶ O compilador não sabe onde está nem *Casa* nem *AreaPiscina*
- ▶ Irá buscar sempre no mesmo diretório do arquivo compilado.
- ▶ Se não estiver lá, dará erro

Linha de Comando

```
$ javac Residencia.java  
Residencia.java:6: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Casa casa;  
    ^  
Residencia.java:8: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    AreaPiscina piscina;  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class Casa  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
    piscina) {  
    ^  
Residencia.java:13: cannot find symbol  
symbol   : class AreaPiscina  
location: class Residencia  
    Residencia(Casa casa, AreaPiscina  
    piscina) {  
    ^  
4 errors
```

Pacotes

- E como dizemos a ele onde está?

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Linha de Comando

```
$ javac Residencia.java
Residencia.java:1: casa.Casa is not public
in casa; cannot be accessed from outside
package
import casa.Casa;
      ^
Residencia.java:2: piscina.AreaPiscina is
not public in piscina; cannot be accessed
from outside package
import piscina.AreaPiscina;
      ^
...
6 errors
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...
 - ▶ Se não especificamos o modificador, ele é “friendly”

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Linha de Comando

```
$ javac Residencia.java
Residencia.java:1: casa.Casa is not public
in casa; cannot be accessed from outside
package
import casa.Casa;
      ^
Residencia.java:2: piscina.AreaPiscina is
not public in piscina; cannot be accessed
from outside package
import piscina.AreaPiscina;
      ^
...
6 errors
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...
 - ▶ Se não especificamos o modificador, ele é “friendly”
 - ★ Acesso somente interno ao pacote

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Linha de Comando

```
$ javac Residencia.java
Residencia.java:1: casa.Casa is not public
in casa; cannot be accessed from outside
package
import casa.Casa;
      ^
Residencia.java:2: piscina.AreaPiscina is
not public in piscina; cannot be accessed
from outside package
import piscina.AreaPiscina;
      ^
...
6 errors
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...
 - ▶ Se não especificamos o modificador, ele é “friendly”
 - ★ Acesso somente interno ao pacote
- Que fazer?

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Linha de Comando

```
$ javac Residencia.java
Residencia.java:1: casa.Casa is not public
in casa; cannot be accessed from outside
package
import casa.Casa;
      ^
Residencia.java:2: piscina.AreaPiscina is
not public in piscina; cannot be accessed
from outside package
import piscina.AreaPiscina;
      ^
...
6 errors
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...
 - ▶ Se não especificamos o modificador, ele é “friendly”
 - ★ Acesso somente interno ao pacote
- Que fazer?
 - ▶ Tornar público o acesso

```
import casa.Casa;
import piscina.AreaPiscina;

class Residencia {
    Casa casa;
    AreaPiscina piscina;

    Residencia(Casa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Linha de Comando

```
$ javac Residencia.java
Residencia.java:1: casa.Casa is not public
in casa; cannot be accessed from outside
package
import casa.Casa;
      ^
Residencia.java:2: piscina.AreaPiscina is
not public in piscina; cannot be accessed
from outside package
import piscina.AreaPiscina;
      ^
...
6 errors
```

Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...
 - ▶ Se não especificamos o modificador, ele é “friendly”
 - ★ Acesso somente interno ao pacote
- Que fazer?
 - ▶ Tornar público o acesso

```
package piscina;  
public class AreaPiscina {... }
```

```
import casa.Casa;  
import piscina.AreaPiscina;
```

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

Linha de Comando

```
$ javac Residencia.java  
Residencia.java:1: casa.Casa is not public  
in casa; cannot be accessed from outside  
package  
import casa.Casa;  
      ^  
Residencia.java:2: piscina.AreaPiscina is  
not public in piscina; cannot be accessed  
from outside package  
import piscina.AreaPiscina;  
      ^  
...  
6 errors
```


Pacotes

- E como dizemos a ele onde está?
 - ▶ Novamente, com *import*
- E...
 - ▶ Se não especificamos o modificador, ele é “friendly”
 - ★ Acesso somente interno ao pacote
- Que fazer?
 - ▶ Tornar público o acesso

```
package piscina;  
public class AreaPiscina {... }
```

e

```
package casa;  
public class Casa {... }
```

```
import casa.Casa;  
import piscina.AreaPiscina;
```

```
class Residencia {  
    Casa casa;  
    AreaPiscina piscina;  
  
    Residencia(Casa casa, AreaPiscina piscina) {  
        this.casa = casa;  
        this.piscina = piscina;  
    }  
}
```

Linha de Comando

```
$ javac Residencia.java  
Residencia.java:1: casa.Casa is not public  
in casa; cannot be accessed from outside  
package  
import casa.Casa;  
      ^  
  
Residencia.java:2: piscina.AreaPiscina is  
not public in piscina; cannot be accessed  
from outside package  
import piscina.AreaPiscina;  
      ^  
...  
6 errors
```

Pacotes

- O mesmo deve ser feito com projeto:

Pacotes

- O mesmo deve ser feito com projeto:

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas
 - ▶ *CasaRet*

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas
 - ▶ *CasaRet*
 - ▶ *CasaQuad*

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas
 - ▶ *CasaRet*
 - ▶ *CasaQuad*
- Bem como

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas
 - ▶ *CasaRet*
 - ▶ *CasaQuad*
- Bem como
 - ▶ O construtor *CasaRet*

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```


Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas
 - ▶ *CasaRet*
 - ▶ *CasaQuad*
- Bem como
 - ▶ O construtor *CasaRet*
 - ▶ O construtor *CasaQuad*

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- O mesmo deve ser feito com projeto:
- Contudo, para que o *main* possa criar os objetos, teremos também que tornar públicas
 - ▶ *CasaRet*
 - ▶ *CasaQuad*
- Bem como
 - ▶ O construtor *CasaRet*
 - ▶ O construtor *CasaQuad*
 - ▶ Os métodos *area* (nas 3 classes de *casa*)

```
import java.util.Scanner;
import casa.CasaRet;
import casa.CasaQuad;
import casa.Casa;
import piscina.AreaPiscina;

class Projeto {
    Residencia[] condominio;
    int ultimo = -1;

    ...

    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);

        System.out.println("Área (r1): "+
                           r1.casa.area());
    }
}
```

Pacotes

- Agora temos todo o projeto pronto.

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?
 - ▶ Funciona porque estamos no mesmo diretório dele

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?
 - ▶ Funciona porque estamos no mesmo diretório dele
- E se quisermos rodar *CasaQuad*, por exemplo, que está dentro de *casa*?

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?
 - ▶ Funciona porque estamos no mesmo diretório dele
- E se quisermos rodar *CasaQuad*, por exemplo, que está dentro de *casa*?
 - ▶ `java casa.CasaQuad` ou

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```


Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?
 - ▶ Funciona porque estamos no mesmo diretório dele
- E se quisermos rodar *CasaQuad*, por exemplo, que está dentro de *casa*?
 - ▶ `java casa.CasaQuad` ou
 - ▶ `java casa/CasaQuad`

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?
 - ▶ Funciona porque estamos no mesmo diretório dele
- E se quisermos rodar *CasaQuad*, por exemplo, que está dentro de *casa*?
 - ▶ `java casa.CasaQuad` ou
 - ▶ `java casa/CasaQuad`

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```

Linha de Comando

```
$ java casa/CasaQuad  
CasaQuad
```

```
$ java casa.CasaQuad  
CasaQuad
```

Pacotes

- Agora temos todo o projeto pronto.
- Como fazemos para rodar *Projeto*?
 - ▶ Funciona porque estamos no mesmo diretório dele
- E se quisermos rodar *CasaQuad*, por exemplo, que está dentro de *casa*?
 - ▶ `java casa.CasaQuad` ou
 - ▶ `java casa/CasaQuad`
 - ▶ Repare que é `"/` e não `"\`

Linha de Comando

```
$ java Projeto  
Área (r1): 150.0
```

Linha de Comando

```
$ java casa/CasaQuad  
CasaQuad
```

```
$ java casa.CasaQuad  
CasaQuad
```

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*

```
package subdir1.subdir2;  
  
public class X {  
    ....  
}
```

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*
 - ▶ Todo o caminho a partir da raiz do projeto – de onde está a classe principal e a partir de onde todas serão compiladas.

```
package subdir1.subdir2;  
  
public class X {  
    ....  
}
```

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*
 - ▶ Todo o caminho a partir da raiz do projeto – de onde está a classe principal e a partir de onde todas serão compiladas.
- E para compilar?

```
package subdir1.subdir2;  
  
public class X {  
    ....  
}
```


Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*
 - ▶ Todo o caminho a partir da raiz do projeto – de onde está a classe principal e a partir de onde todas serão compiladas.
- E para compilar?

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*
 - ▶ Todo o caminho a partir da raiz do projeto – de onde está a classe principal e a partir de onde todas serão compiladas.
- E para compilar?
- E para rodar?

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Pacotes

- Finalmente, e se a classe estiver em um subdiretório dentro de um subdiretório?
 - ▶ Colocamos todo o caminho na declaração *package*
 - ▶ Todo o caminho a partir da raiz do projeto – de onde está a classe principal e a partir de onde todas serão compiladas.
- E para compilar?
- E para rodar?

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Linha de Comando

```
$ java subdir1/subdir2/X
```

```
$ java subdir1.subdir2.X
```

Pacotes

- Repare que Java sempre pressupõe uma hierarquia de pacotes

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Linha de Comando

```
$ java subdir1/subdir2/X
```

```
$ java subdir1.subdir2.X
```

Pacotes

- Repare que Java sempre pressupõe uma hierarquia de pacotes
 - ▶ Tudo deve ser planejado de modo que haja uma raiz

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Linha de Comando

```
$ java subdir1/subdir2/X
```

```
$ java subdir1.subdir2.X
```

Pacotes

- Repare que Java sempre pressupõe uma hierarquia de pacotes
 - ▶ Tudo deve ser planejado de modo que haja uma raiz
 - ▶ Tudo deve ser compilado/rodado a partir dessa raiz

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Linha de Comando

```
$ java subdir1/subdir2/X
```

```
$ java subdir1.subdir2.X
```

Pacotes

- Repare que Java sempre pressupõe uma hierarquia de pacotes
 - ▶ Tudo deve ser planejado de modo que haja uma raiz
 - ▶ Tudo deve ser compilado/rodado a partir dessa raiz
 - ▶ Não há um ../

```
package subdir1.subdir2;
```

```
public class X {  
    ....  
}
```

Linha de Comando

```
$ javac subdir1/subdir2/X.java
```

Linha de Comando

```
$ java subdir1/subdir2/X
```

```
$ java subdir1.subdir2.X
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?

```
public class X {  
    private int y = 2;  
  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
    }  
}
```


Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:

```
public class X {  
    private int y = 2;  
  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(obj.y);  
    }  
}
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:
- Que fazer?

```
public class X {  
    private int y = 2;
```

```
}
```

```
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(obj.y);  
    }  
}
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:
- Que fazer?
 - ▶ Dar acesso a ele via um método público: um getter

```
public class X {  
    private int y = 2;  
  
    public int getY() {  
        return(y);  
    }  
  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(obj.y);  
    }  
}
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:
- Que fazer?
 - ▶ Dar acesso a ele via um método público: um getter
 - ▶ E então fazer uma chamada a esse método

```
public class X {  
    private int y = 2;  
  
    public int getY() {  
        return(y);  
    }  
  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(  
            obj.getY());  
    }  
}
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:
- Que fazer?
 - ▶ Dar acesso a ele via um método público: um getter
 - ▶ E então fazer uma chamada a esse método
- Da mesma forma, seu valor pode ser mudado por outro método público

```
public class X {  
    private int y = 2;  
  
    public int getY() {  
        return(y);  
    }  
  
    public void setY(int v) {  
        y = v;  
    }  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(  
            obj.getY());  
    }  
}
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:
- Que fazer?
 - ▶ Dar acesso a ele via um método público: um getter
 - ▶ E então fazer uma chamada a esse método
- Da mesma forma, seu valor pode ser mudado por outro método público
 - ▶ Um setter

```
public class X {  
    private int y = 2;  
  
    public int getY() {  
        return(y);  
    }  
  
    public void setY(int v) {  
        y = v;  
    }  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(  
            obj.getY());  
    }  
}
```

Getters e Setters

- E se desejássemos permitir o acesso indireto a um atributo privado, a partir de outra classe?
 - ▶ Não poderíamos fazer:
- Que fazer?
 - ▶ Dar acesso a ele via um método público: um getter
 - ▶ E então fazer uma chamada a esse método
- Da mesma forma, seu valor pode ser mudado por outro método público
 - ▶ Um setter
 - ▶ Usado da mesma forma

```
public class X {  
    private int y = 2;  
  
    public int getY() {  
        return(y);  
    }  
  
    public void setY(int v) {  
        y = v;  
    }  
}  
  
public class Y {  
    public void f() {  
        X obj = new X();  
        System.out.println(  
            obj.getY());  
        obj.setY(3);  
    }  
}
```

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>				
<i>protected</i>				
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S			
<i>protected</i>				
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S		
<i>protected</i>				
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	
<i>protected</i>				
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>				
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S			
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S		
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>				
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S			
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S		
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S	N	
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S	N	N
<i>private</i>				

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S	N	N
<i>private</i>	S			

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S	N	N
<i>private</i>	S	N		

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S	N	N
<i>private</i>	S	N	N	

Modificadores

- Quanto ao acesso:

<i>Modificador</i>	<i>Mesma Classe</i>	<i>Mesmo Pacote</i>	<i>Subclasse</i>	<i>Outros Pacotes</i>
<i>public</i>	S	S	S	S
<i>protected</i>	S	S	S	N
<i>nada</i>	S	S	N	N
<i>private</i>	S	N	N	N

Modificadores

- Outros modificadores:

Modificadores

- Outros modificadores:
 - ▶ *final classe, atributo ou método:*

Modificadores

- Outros modificadores:
 - ▶ *final classe, atributo ou método:*
 - ★ Classe: a classe não aceita subclasses

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método:*

- ★ Classe: a classe não aceita subclasses

- ★ Atributo: seu valor não pode ser mudado (é constante)

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método:*

- ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
- ▶ *static atributo ou método*: cópia única na memória

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
- ▶ *static atributo ou método*: cópia única na memória
 - ★ Se usados em classes internas, as torna externas

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método:*

- ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito

- ▶ *static atributo ou método:* cópia única na memória

- ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
- ▶ *static atributo ou método*: cópia única na memória
 - ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto
- ▶ *abstract*: veremos mais adiante

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
- ▶ *static atributo ou método*: cópia única na memória
 - ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto
- ▶ *abstract*: veremos mais adiante
- ▶ *native, strictfp, synchronized, transient, volatile*

Modificadores

- Outros modificadores:
 - ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
 - ▶ *static atributo ou método*: cópia única na memória
 - ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto
 - ▶ *abstract*: veremos mais adiante
 - ▶ *native, strictfp, synchronized, transient, volatile*
- Podemos também misturar alguns. Ex:

Modificadores

- Outros modificadores:
 - ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
 - ▶ *static atributo ou método*: cópia única na memória
 - ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto
 - ▶ *abstract*: veremos mais adiante
 - ▶ *native, strictfp, synchronized, transient, volatile*
- Podemos também misturar alguns. Ex:
 - ▶ *public static final*

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método:*

- ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito

- ▶ *static atributo ou método:* cópia única na memória

- ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto

- ▶ *abstract:* veremos mais adiante

- ▶ *native, strictfp, synchronized, transient, volatile*

- Podemos também misturar alguns. Ex:

- ▶ *public static final*

- ▶ *static abstract final*

Modificadores

- Outros modificadores:

- ▶ *final classe, atributo ou método*:
 - ★ Classe: a classe não aceita subclasses
 - ★ Atributo: seu valor não pode ser mudado (é constante)
 - ★ Método: não pode ser sobrescrito
- ▶ *static atributo ou método*: cópia única na memória
 - ★ Se usados em classes internas, as torna externas
 - ★ Indica quais atributos/métodos devem ser considerados pertencentes à classe e não específicos a cada objeto
- ▶ *abstract*: veremos mais adiante
- ▶ *native, strictfp, synchronized, transient, volatile*

- Podemos também misturar alguns. Ex:

- ▶ *public static final*
- ▶ *static abstract final*
- ▶ etc