Aula 20 – Especificadores de acesso e Pacotes

Norton Trevisan Roman

11 de junho de 2018

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() {
    return(this.lateral>=0 ?
          this.lateral*this.lateral : -1);
  }
}
```

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() {
    return(this.lateral>=0 ?
          this.lateral*this.lateral : -1);
  }
}
```

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() {
    return(this.lateral>=0 ?
          this.lateral*this.lateral : -1);
  }
}
```

Private

```
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?

Private

```
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
```

Private

```
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);
  }
}
```

- Não temos acesso ao valor do m^2

Private

```
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);
  }
}
```

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

Private

```
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);
  }
}
```

- Que fazer?

Private

```
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);
  }
}
```

- Que fazer?
- Refatorar a classe

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 (Modificadores) de Acesso

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

Especificadores (Modificadores) de Acesso

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

Especificadores (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 (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 (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 (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 (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 (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 (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

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
    - Nesse caso, a primeira seria `normal/Matematica.class` e a segunda em `complexo/Matematica.class`

Pacotes

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

Pacotes

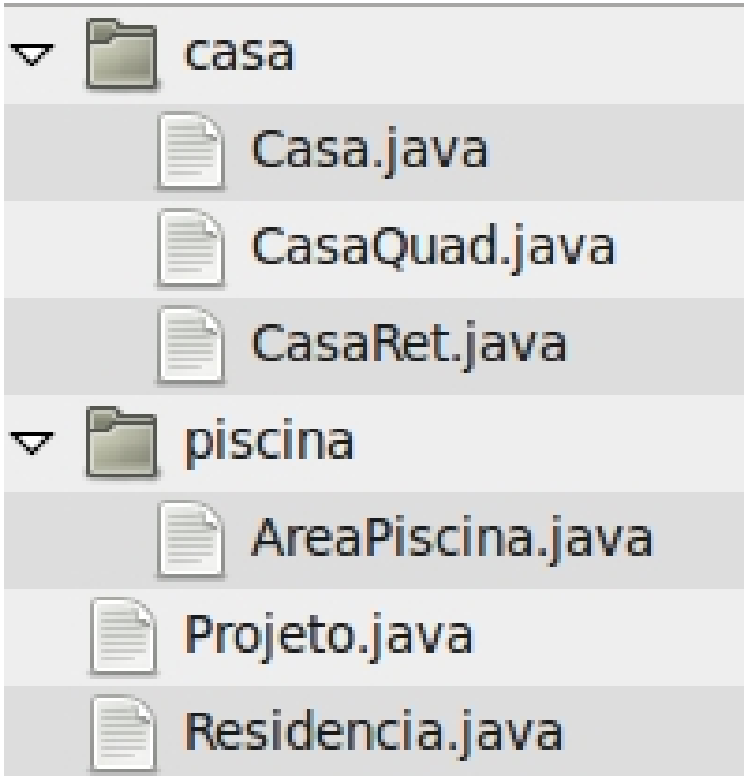
- Que agrupamento (pacotes) podemos fazer em nossas classes?
- Poderíamos separar as coisas relativas à casa das relativas à piscina

Pacotes

- Que agrupamento (pacotes) podemos fazer em nossas classes?
- Poderíamos separar as coisas relativas à casa das relativas à piscina
  - 2 pacotes: `casa` e `piscina`

Pacotes

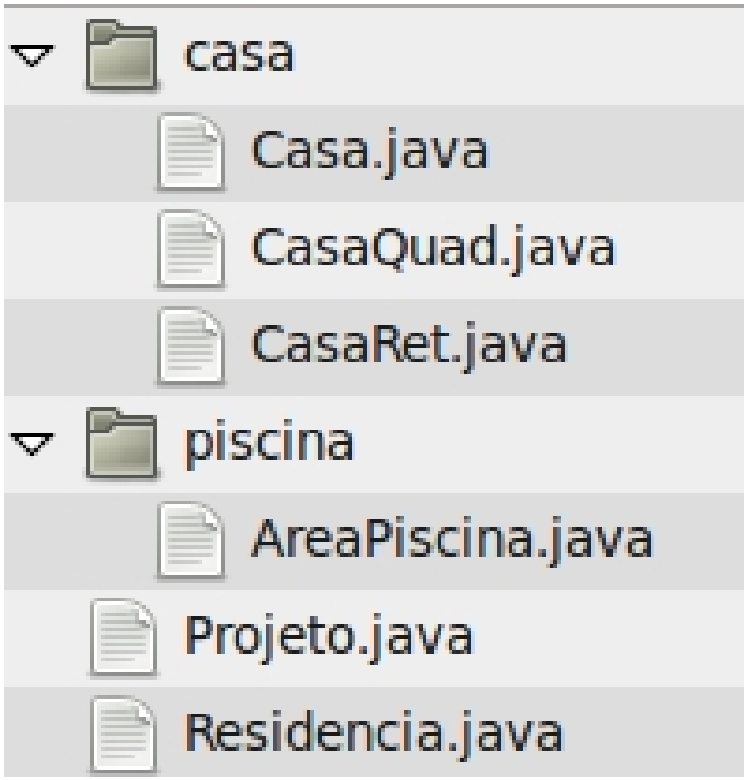
- Que agrupamento (pacotes) podemos fazer em nossas classes?
- Poderíamos separar as coisas relativas à casa das relativas à piscina
  - 2 pacotes: `casa` e `piscina`



Pacotes

Casa.java:

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



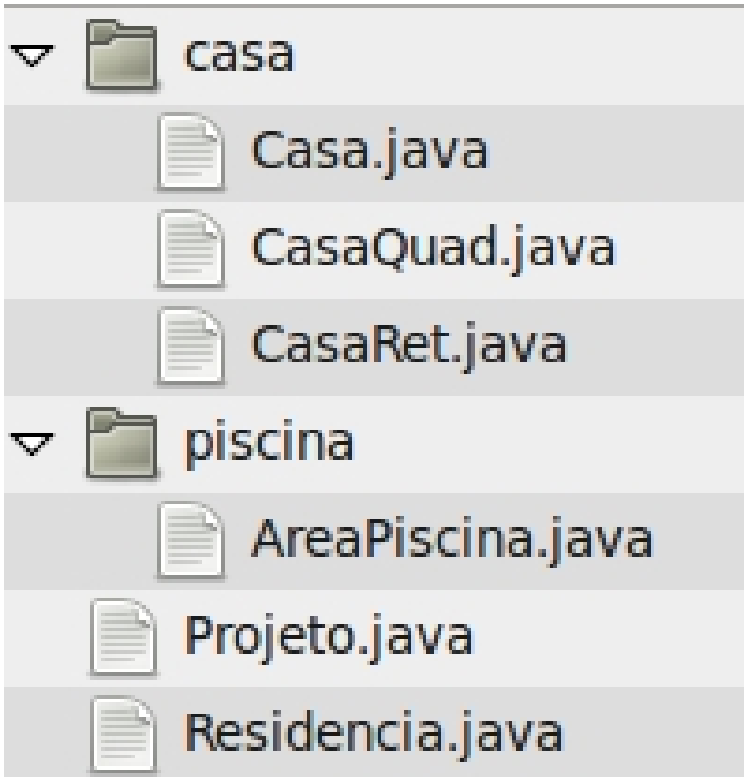
Pacotes

Casa.java:

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

CasaQuad.java:

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



Pacotes

Casa.java:

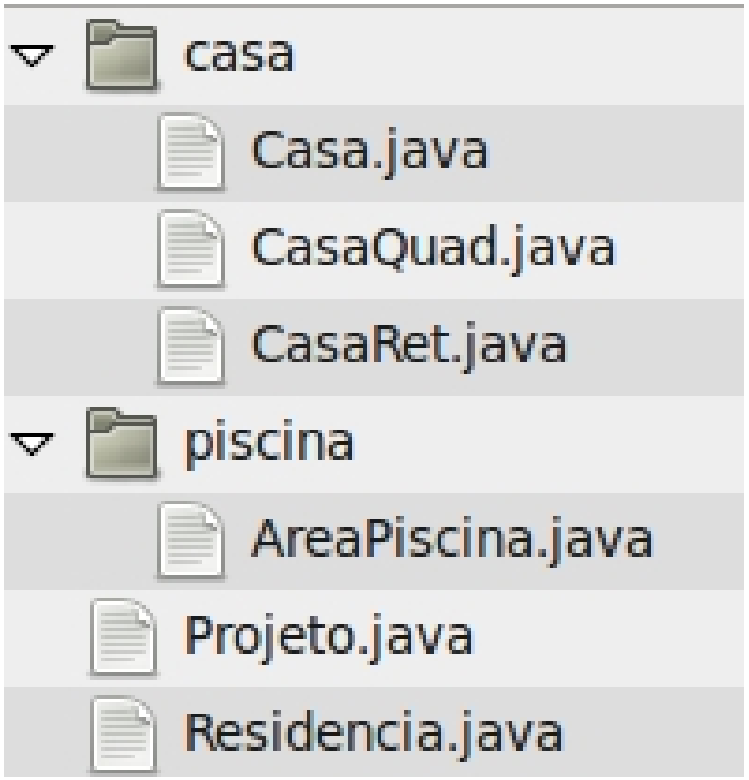
```
package casa;
class Casa { ... }
```

CasaQuad.java:

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

CasaRet.java:

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



Pacotes

Casa.java:

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

CasaQuad.java:

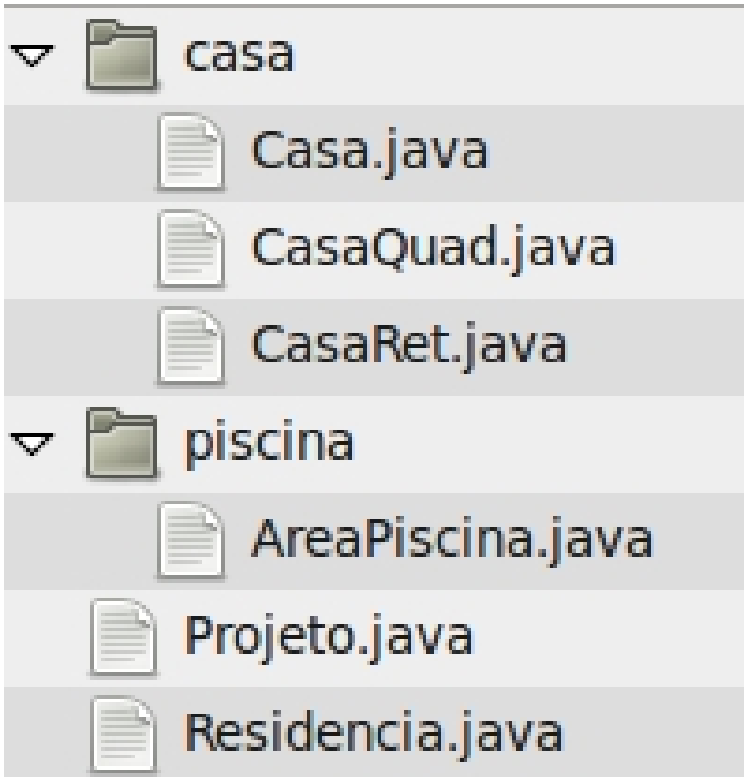
```
package casa;
class CasaQuad extends Casa { .. }
```

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 pai, onde estão os diretórios `casa` e `piscina`

Pacotes

- E como compilamos isso?
  - Do diretório pai, onde estão os diretórios `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 pai, onde estão os diretórios `casa` e `piscina`

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

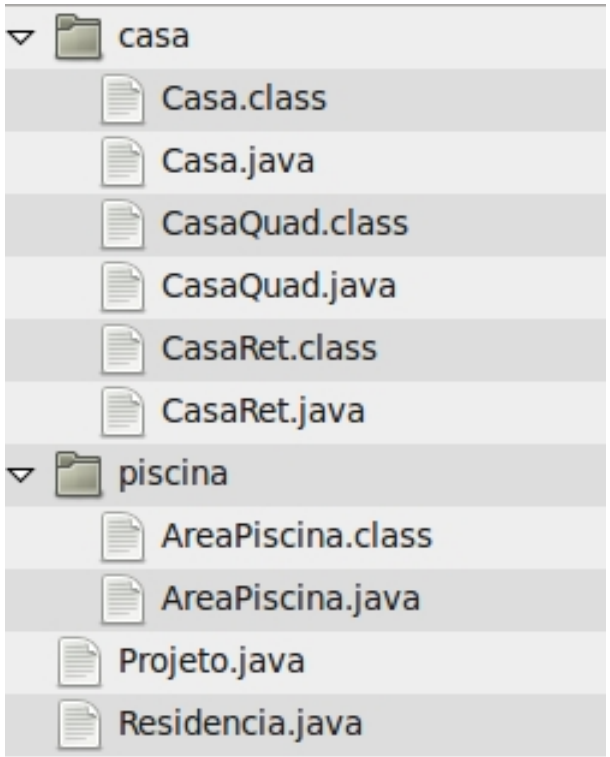
- E o diretório conterá

Pacotes

- E como compilamos isso?
  - Do diretório pai, onde estão os diretórios `casa` e `piscina`

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

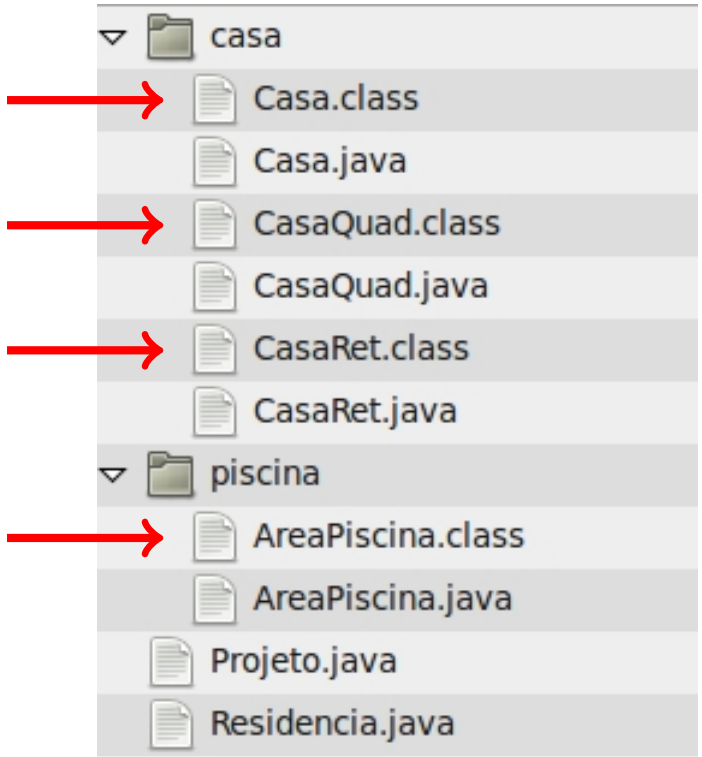
- E o diretório conterá



Pacotes

- E como compilamos isso?
  - Do diretório pai, onde estão os diretórios `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

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;
  }
}
```

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

- 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

- 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

- 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`

```
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

- Se não especificamos o modificador, ele é "friendly"

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

- Se não especificamos o modificador, ele é "friendly"
  - Acesso somente interno ao pacote

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

- Se não especificamos o modificador, ele é "friendly"
  - Acesso somente interno ao pacote
- Que fazer?

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

- Tornar público o acesso

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

- Tornar público o acesso

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

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

- Tornar público o acesso

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

e

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

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

- 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

- 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

- 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

- Bem como
  - O construtor `CasaRet`
  - O construtor `CasaQuad`
  - Os métodos `area` (nas 3 classes do diretório `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`?

Linha de Comando

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

- Funciona porque estamos no mesmo diretório dele

Pacotes

- E se quisermos rodar `CasaQuad`, por exemplo, que está dentro do diretório `casa`?

Pacotes

- E se quisermos rodar `CasaQuad`, por exemplo, que está dentro do diretório `casa`?
  - `java casa.CasaQuad` ou

Pacotes

- E se quisermos rodar `CasaQuad`, por exemplo, que está dentro do diretório `casa`?
  - `java casa.CasaQuad` ou
  - `java casa/CasaQuad`

Pacotes

- E se quisermos rodar `CasaQuad`, por exemplo, que está dentro do diretório `casa`?
  - `java casa.CasaQuad` ou
  - `java casa/CasaQuad`

Linha de Comando

```
$ java casa/CasaQuad
CasaQuad

$ java casa.CasaQuad
CasaQuad
```

(em seu `main` há somente um `println` com o nome da classe)

Pacotes

- E se quisermos rodar `CasaQuad`, por exemplo, que está dentro do diretório `casa`?
  - `java casa.CasaQuad` ou
  - `java casa/CasaQuad`
  - Repare que é / e não \

Linha de Comando

```
$ java casa/CasaQuad
CasaQuad

$ java casa.CasaQuad
CasaQuad
```

(em seu `main` há somente um `println` com o nome da classe)

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`

```
package subdir1.subdir2;

public class X {
   ...
}
```

  - Todo o caminho a partir da raiz do projeto → onde está a classe principal e a partir de onde todas serão compiladas.

Pacotes

- E para compilar?

Pacotes

- E para compilar?

Linha de Comando

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

Pacotes

- E para compilar?

Linha de Comando

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

- E para rodar?

Pacotes

- E para compilar?

Linha de Comando

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

- E para rodar?

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

- 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

- 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

- 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);
  }
}
```

Getters e Setters

Possíveis usos:

```
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);
  }
}
```

Getters e Setters

Possíveis usos:

- Permitir acesso apenas para escrita (remove-se o getter)

```
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);
  }
}
```

Getters e Setters

Possíveis usos:

- Permitir acesso apenas para escrita (remove-se o getter)
- Permitir acesso apenas para leitura (remove-se o 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());
    obj.setY(3);
  }
}
```

Getters e Setters

Possíveis usos:

- Permitir acesso apenas para escrita (remove-se o getter)
- Permitir acesso apenas para leitura (remove-se o setter)
- Testar parâmetro antes de abastecer atributo (dentro do 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());
    obj.setY(3);
  }
}
```

Getters e Setters

Possíveis usos:

- Mudar o formato de um atributo antes de retornar (dentro do getter)

```
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);
  }
}
```

Getters e Setters

Possíveis usos:

- Mudar o formato de um atributo antes de retornar (dentro do getter)
  - Ex: Um `double` 323 que vira o `String` "R$ 323,00"

```
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:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public				
protected				
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S			
protected				
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S		
protected				
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	
protected				
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected				
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S			
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S		
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada				
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S			
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S		
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S	N	
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S	N	N
private				

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S	N	N
private	S			

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S	N	N
private	S	N		

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S	N	N
private	S	N	N	

Modificadores

Quanto ao acesso:

Modificador	*Mesma Classe*	*Mesmo Pacote*	*Subclasse*	*Outros Pacotes*
public	S	S	S	S
protected	S	S	S	N
nada	S	S	N	N
private	S	N	N	N

Modificadores

Outros modificadores:

Modificadores

Outros modificadores:

- `final` *atributo, método ou classe*:

Modificadores

Outros modificadores:

- `final` *atributo, método ou classe*:
  - Atributo: seu valor não pode ser mudado (é constante)

Modificadores

Outros modificadores:

- `final` *atributo, método ou classe*:
  - Atributo: seu valor não pode ser mudado (é constante)
  - Método: não pode ser sobrescrito

Modificadores

Outros modificadores:

- `final` *atributo, método ou classe*:
  - Atributo: seu valor não pode ser mudado (é constante)
  - Método: não pode ser sobrescrito
  - Classe: a classe não aceita subclasses

Modificadores

Outros modificadores:

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

Modificadores

Outros modificadores:

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

Modificadores

Outros modificadores:

- `final` *atributo, método ou classe*:
  - Atributo: seu valor não pode ser mudado (é constante)
  - Método: não pode ser sobrescrito
  - Classe: a classe não aceita subclasses
- `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:

- `abstract`: veremos mais adiante

Modificadores

Outros modificadores:

- `abstract`: veremos mais adiante
- `native`, `strictfp`, `synchronized`, `trasient`, `volatile`: não veremos

Modificadores

Outros modificadores:

- `abstract`: veremos mais adiante
- `native`, `strictfp`, `synchronized`, `trasient`, `volatile`: não veremos
- Podemos também misturar alguns. Ex:

Modificadores

Outros modificadores:

- `abstract`: veremos mais adiante
- `native`, `strictfp`, `synchronized`, `trasient`, `volatile`: não veremos
- Podemos também misturar alguns. Ex:
  - `public static final`

Modificadores

Outros modificadores:

- `abstract`: veremos mais adiante
- `native`, `strictfp`, `synchronized`, `trasient`, `volatile`: não veremos
- Podemos também misturar alguns. Ex:
  - `public static final`
  - `static abstract final`

Modificadores

Outros modificadores:

- `abstract`: veremos mais adiante
- `native`, `strictfp`, `synchronized`, `trasient`, `volatile`: não veremos
- Podemos também misturar alguns. Ex:
  - `public static final`
  - `static abstract final`
  - etc.

Construtores Private

Possível uso: Singleton

```
public class Singleton {

  private static Singleton
          copia = new Singleton();

  private Singleton() {}

  public static Singleton copia()
  {
    return copia;
  }
}
```

Construtores Private

Possível uso: Singleton

- Garante a existência de cópia única do objeto

```
public class Singleton {

  private static Singleton
          copia = new Singleton();

  private Singleton() {}

  public static Singleton copia()
  {
    return copia;
  }
}
```

Construtores Private

Possível uso: Singleton

- Garante a existência de cópia única do objeto
  - Ninguém consegue criar objeto da classe

```
public class Singleton {

  private static Singleton
          copia = new Singleton();

  private Singleton() {}

  public static Singleton copia()
  {
    return copia;
  }
}
```

Construtores Private

Possível uso: Singleton

- Garante a existência de cópia única do objeto
  - Ninguém consegue criar objeto da classe
  - No máximo, pedir a referência à única cópia existente

```
public class Singleton {

  private static Singleton
          copia = new Singleton();

  private Singleton() {}

  public static Singleton copia()
  {
    return copia;
  }
}
```

Videoaula

Não há.