DESENVOLVIMENTO DE SOFTWARE EM LARGA ESCALA

CONCEITOS FUNDAMENTAIS

- ☐ MÓDULOS COMO ABSTRACÇÃO DE DADOS
- ☐ "DATA HIDING"
- ☐ "IMPLEMENTATION HIDING"
- ☐ ENCAPSULAMENTO
- ☐ INDEPENDÊNCIA CONTEXTUAL

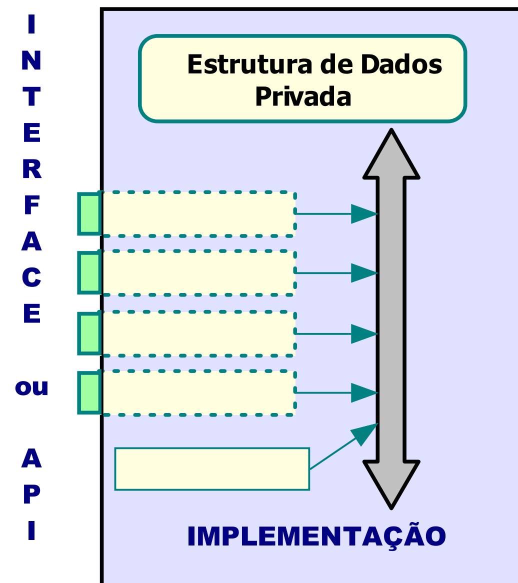
METODOLOGIAS DE DESENHO E DE

PROGRAMAÇÃO ORIENTADAS AOS OBJECTOS

MÓDULO COMO ABSTRACÇÃO DE DADOS
MÓDULO COMO CÁPSULA DE DADOS
ENCAPSULAMENTO E INDEPENDÊNCIA DO CONTEXTO

Módulo = Abstracção de Dados

Módulo = Interface + Implementação



- **API:** único acesso do exterior (Abstracção de Dados)
- **Representação Invisível:** não acessível do exterior (Abstracção de Dados)
- **Robustez:** Erros resultam apenas do código interno, o único que acede a dados
- **Reutilização:** estes módulos são independentes do contexto

Nota: Compiladores não conhecem estas regras; Nós é que temos que as impor.

© F. Mário Martins 2007

METODOLOGIA – SÍNTESE:

1. ***CRIAR OS MÓDULOS ("CÁPSULAS") PENSANDO SOBRE QUAL É O TIPO DE DADOS QUE VAMOS REPRESENTAR E MANIPULAR NO MÓDULO (exº uma STACK, uma TURMA, um PONTO, etc.);***

2. ***DEFINIR TODAS AS VARIÁVEIS (ESTRUTURA DE DADOS) DE REPRESENTAÇÃO PROCURANDO GARANTIR QUE SÃO DADOS PRIVADOS, ISTO É, APENAS ACESSÍVEIS DE DENTRO DO MÓDULO E INACESSÍVEIS DO SEU EXTERIOR;***

3. ***DEFINIR TODAS AS OPERAÇÕES DE ACESSO E DE MODIFICAÇÃO DA REPRESENTAÇÃO INTERNA, E DECIDIR QUAIS SÃO IMPORTANTES PARA SEREM TORNADAS PÚBLICAS (API);***

4. ***NOTAR A IMPORTÂNCIA DE SEREM CRIADAS OPERAÇÕES QUE CONSULTEM O ESTADO INTERNO DA ESTRUTURA DE DADOS; SE O MÓDULO É "OPACO", AS OPERAÇÕES DE CONSULTA SÃO AS ÚNICAS QUE, NO EXTERIOR, PERMITIRÃO SABER OS VALORES DOS DADOS PRIVADOS INTERNOS;***

5. ***NUNCA ESCREVER OPERAÇÕES DE INPUT OU DE OUTPUT NO CÓDIGO DAS OPERAÇÕES, SEJAM ESTAS PÚBLICAS OU NÃO; O I/O TORNA O CÓDIGO DO MÓDULO DEPENDENTE DE DISPOSITIVOS FÍSICOS DE INPUT e OUTPUT;***

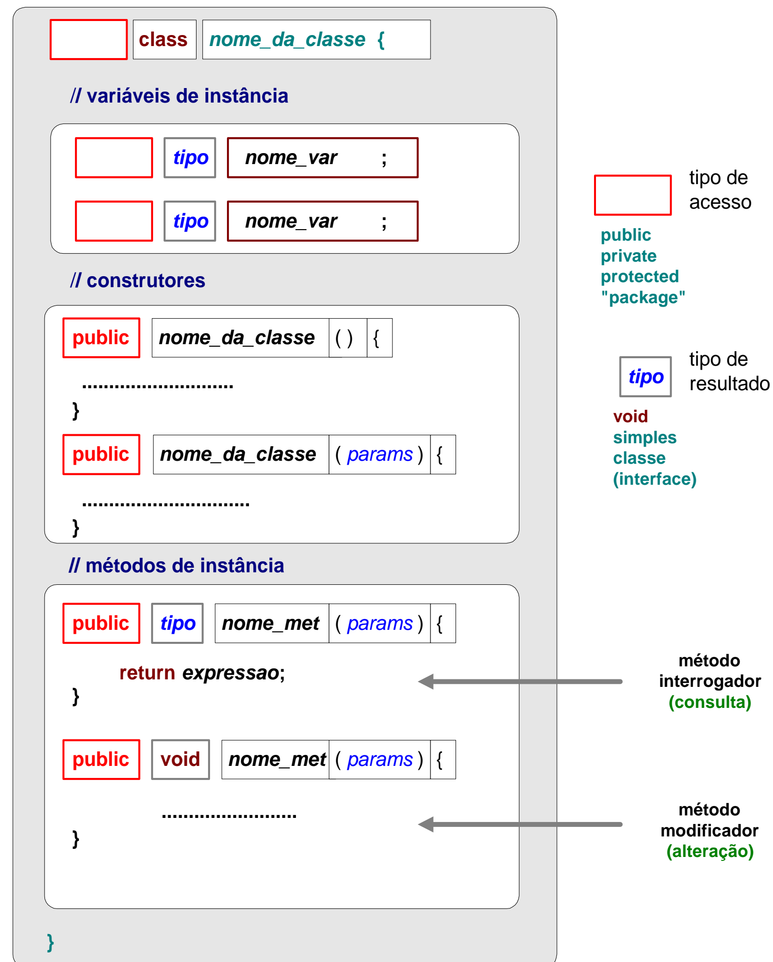
6. ***AO UTILIZAR ESTES MÓDULOS DE DADOS, NUNCA ACEDER DIRECTAMENTE À SUA REPRE-SENTAÇÃO INTERNA (FAZER ABSTRACÇÃO) E USAR APENAS A API.***

© F. Mário Martins 2007

OBJECTOS (COMO MÓDULOS DE DADOS)

- IDENTIDADE ÚNICA
- ESTADO INTERNO: ATRIBUTOS PRIVADOS (VARIÁVEIS DE INSTÂNCIA)
- COMPORTAMENTO: OPERAÇÕES QUE ACEDEM AO ESTADO (MÉTODOS DE INSTÂNCIA)

CLASSES ESPECIFICAM A ESTRUTURA E O COMPORTAMENTO DE CADA TIPO DE OBJECTOS



tipo de acesso

public
private
protected
"package"

tipo

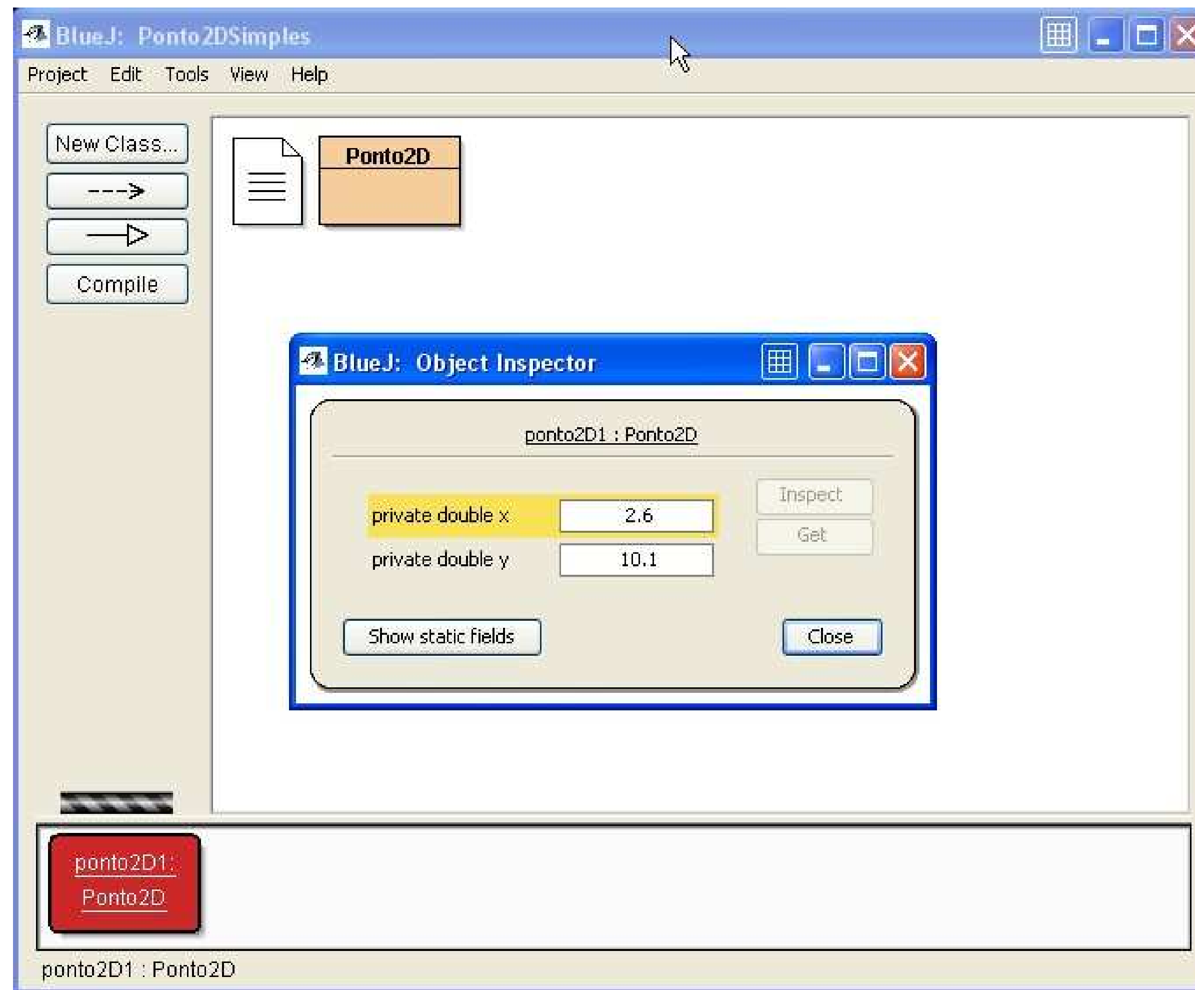
tipo de resultado

void
simples
classe
(interface)

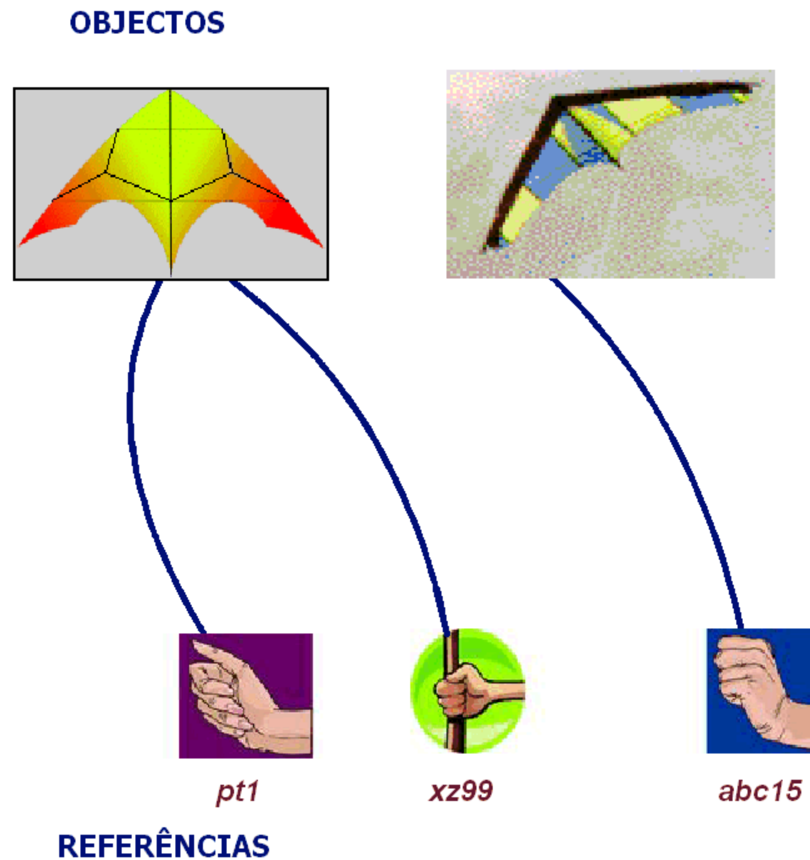
método interrogador (consulta)

método modificador (alteração)

Distinção entre Classes e Instâncias é clara (Pelo menos em BlueJ)



OBJECTOS SÃO DE TIPOS REFERENCIADOS, OU SEJA, AS VARIÁVEIS QUE OS IDENTIFICAM SÃO APENAS "HANDLERS", IE. VARIÁVEIS QUE GUARDAM OS SEUS EFECTIVOS ENDEREÇOS



PEDIDO DE COMPORTAMENTO POR MENSAGENS

```
pt1.descer(10);
abc15.abrir(12);
xz99.direita(2);
pt1.direita(55);
```

ENCAPSULAMENTO EM POO

REGRA 1:

TODAS AS VARIÁVEIS DE INSTÂNCIA, SEJAM DE TIPOS SIMPLES OU DE TIPOS REFERENCIADOS (OBJECTOS), DEVEM SER DECLARADAS COMO `private`, SENDO APENAS ACESSÍVEIS AOS MÉTODOS DE INSTÂNCIA.

```
import static java.lang.Math.abs;
public class Ponto2D {

  // Construtores usuais
  public Ponto2D(double cx, double cy) { x = cx; y = cy; }
  public Ponto2D(){ this(0.0, 0.0); } // usa o outro construtor
  public Ponto2D(Ponto2D p) { x = p.getX(); y = p.getY(); }

  // Variáveis de Instância
  private double x, y;

```

```

import static java.lang.Math.PI;
public class Circulo {
     
  // Variáveis de Instância
  
  private double raio;            // raio
  private Ponto2D centro;         // centro
 
  // Construtores de circulos

  public Circulo() { centro = new Ponto2D(); raio = 1.0; }

```

ENCAPSULAMENTO, PARTILHA DE ENDEREÇOS E CLONING

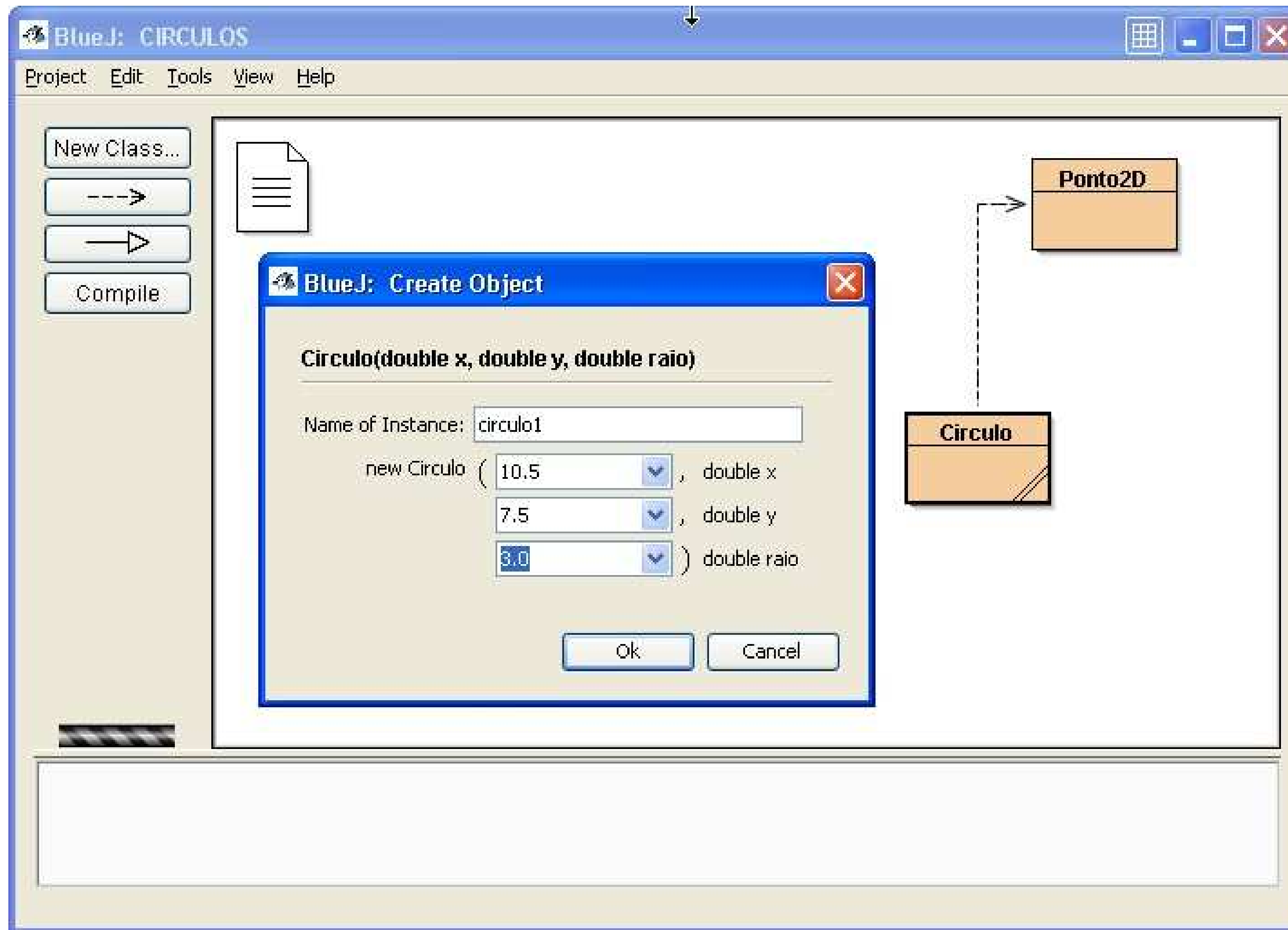
TEOREMA 1:

QUANDO UM MÉTODO DE INSTÂNCIA DÁ COMO RESULTADO UMA DAS VARIÁVEIS DE INSTÂNCIA DE TIPO REFERENCIADO (OU SEJA NÃO É DE TIPO SIMPLES), ESTAMOS A DEVOLVER, DE FACTO, O ENDEREÇO (E NÃO UMA CÓPIA). QUEM RECEBER ESSE ENDEREÇO PODE ALTERAR O ESTADO INTERNO DESSE OBJECTO E ASSIM MODIFICAR O ESTADO INTERNO DO OBJECTO QUE POSSUI TAL VARIÁVEL DE INSTÂNCIA.

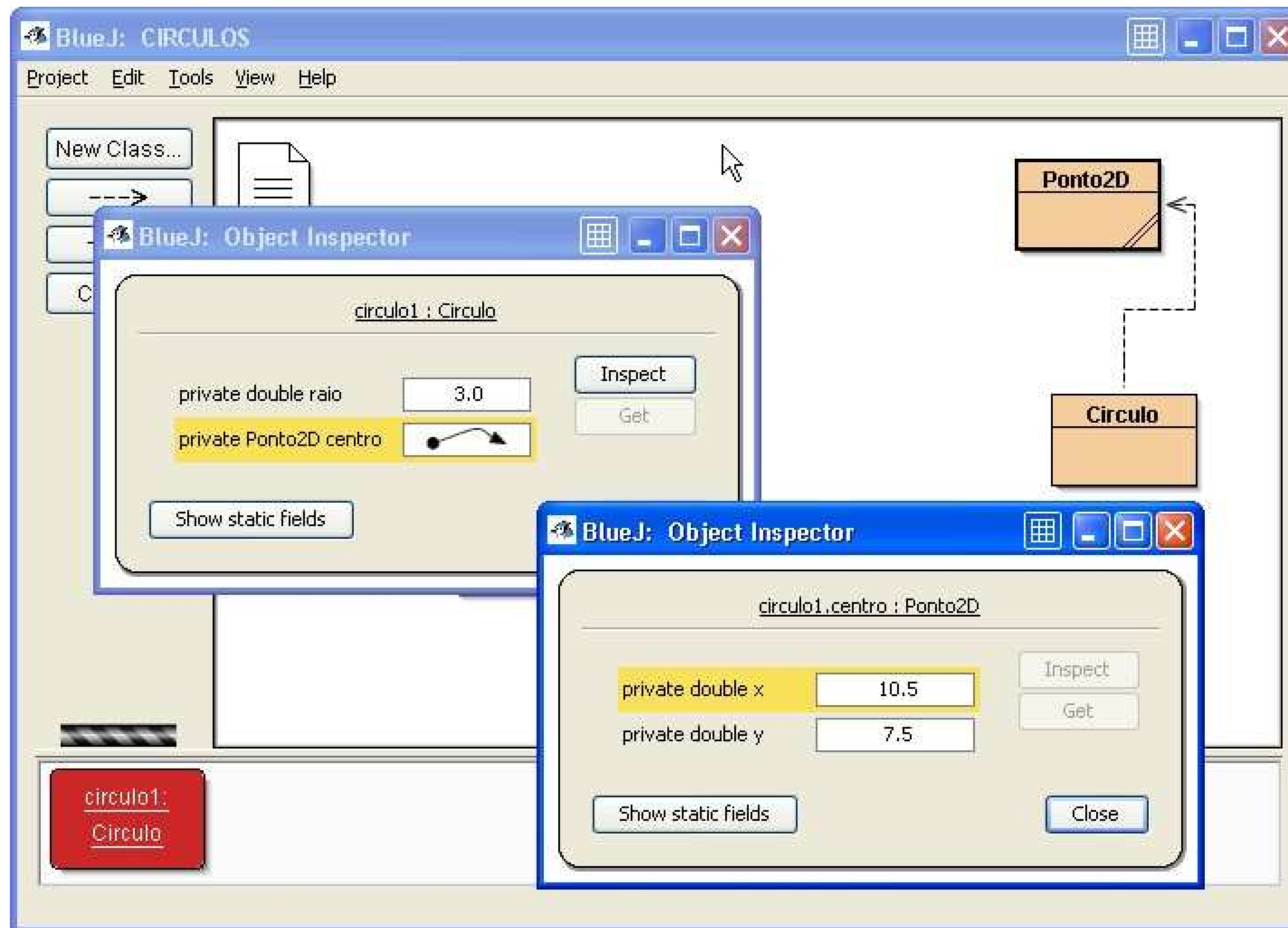
EXEMPLO – CLASSE CIRCULO:

```
import static java.lang.Math.PI;
public class Circulo {

   private double raio;     // o raio do círculo
   private Ponto2D centro;  // ponto que define o centro do círculo

   // Construtores de circulos
   ………………
   // Métodos de instância

   /** Devolve o valor do RAIO. */
   public double getRaio() {return raio;}

    /**
    * Devolve o ponto que representa o CENTRO. Nesta implementação, o Ponto2D devolvido não é
    * uma cópia do CENTRO mas o próprio centro deste círculo. Assim, se for alterado por quem invocou
    * este método e recebeu tal resultado, este círculo terá o seu CENTRO modificado (sem sequer saber
    * porquê nem por quem !!
    */
   public Ponto2D daCentro() {return centro;} // não é uma cópia

…….
}
```

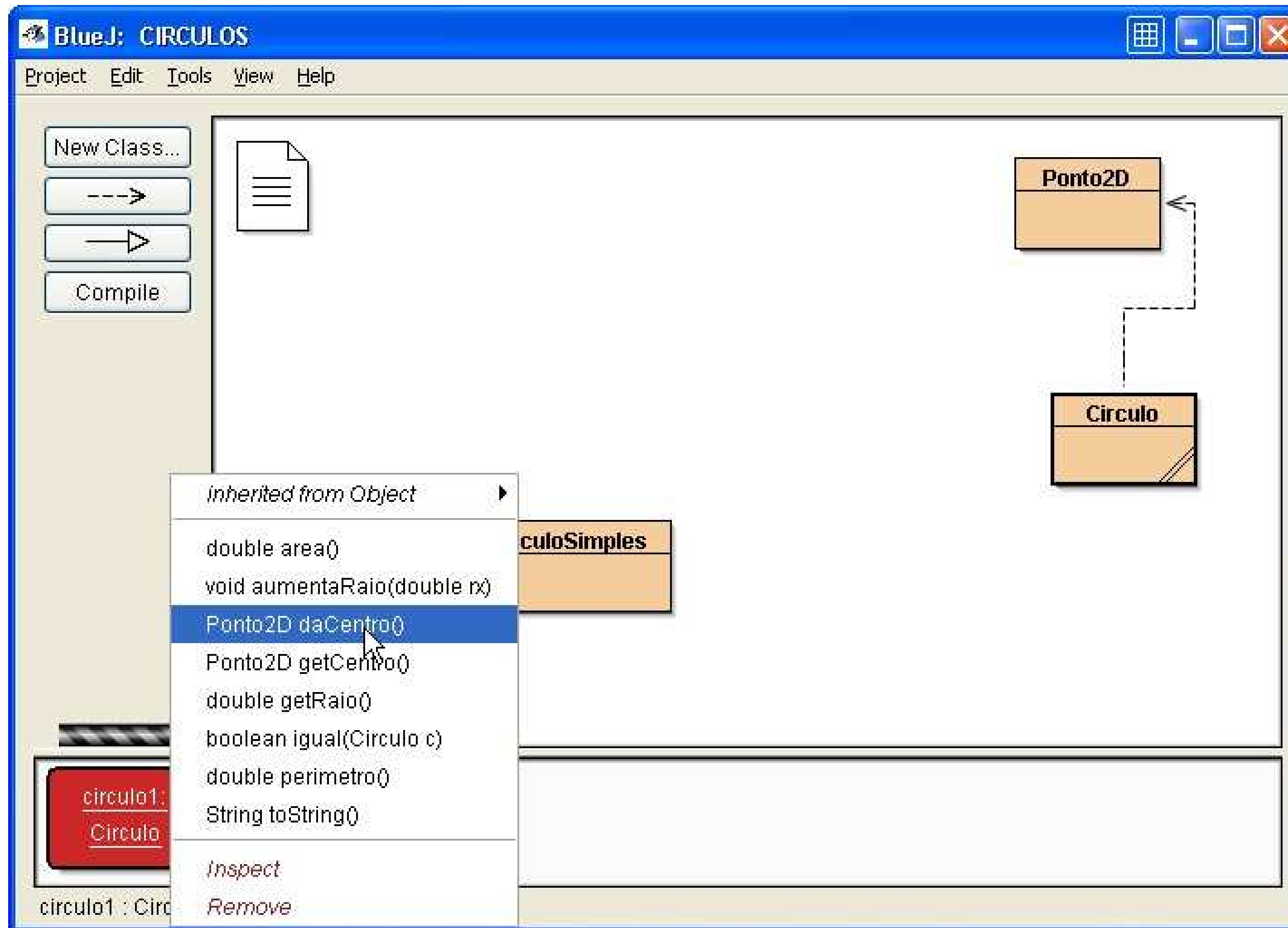
CRIAÇÃO DE UMA INSTÂNCIA DE CIRCULO



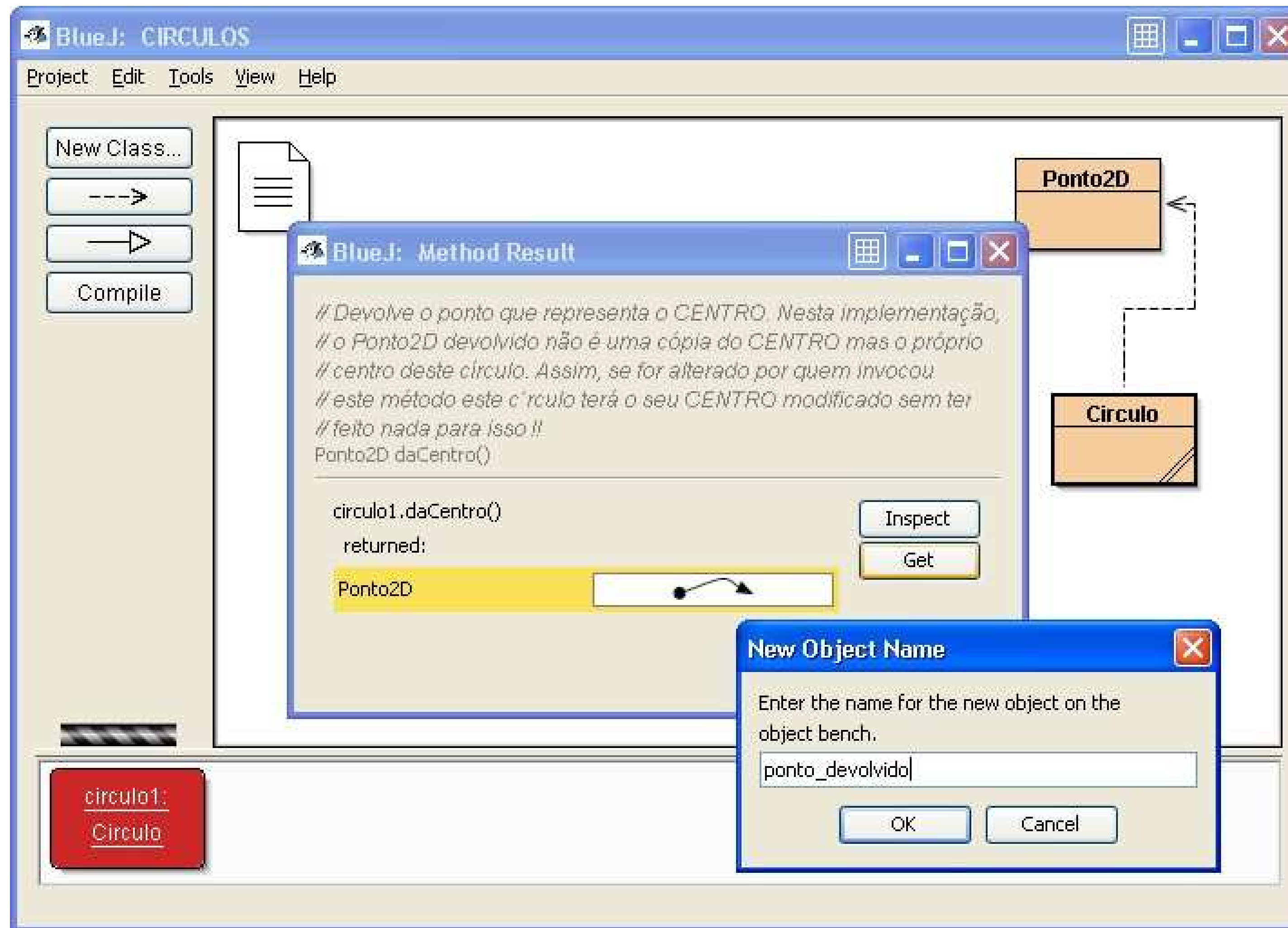
INSPECT



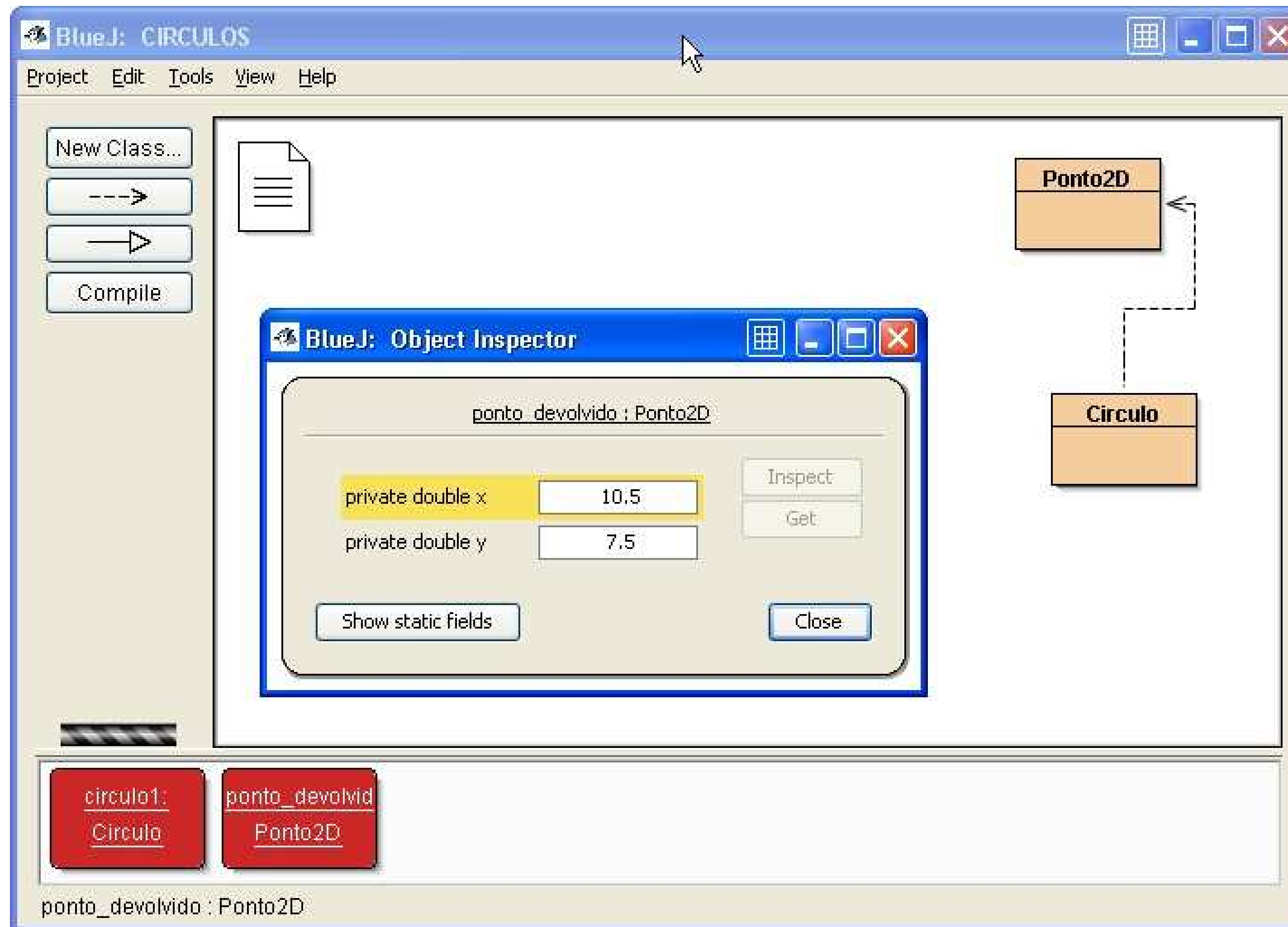
INVOCAÇÃO DO MÉTODO `public Ponto2D daCentro() { return centro; }`



GET DO PONTO RESULTADO



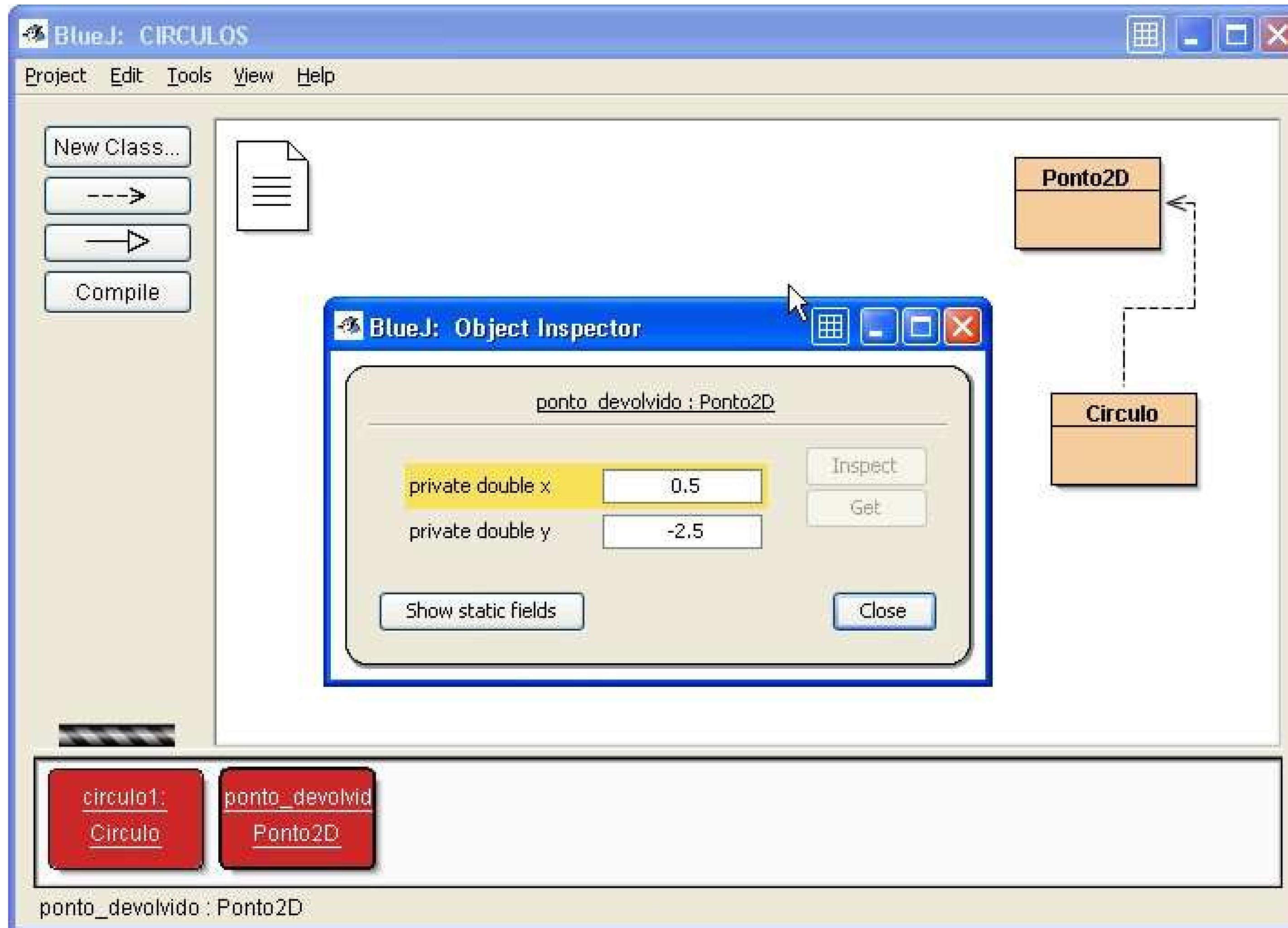
INSPECT DO PONTO RESULTADO (TEM O MESMO VALOR DE CENTRO)



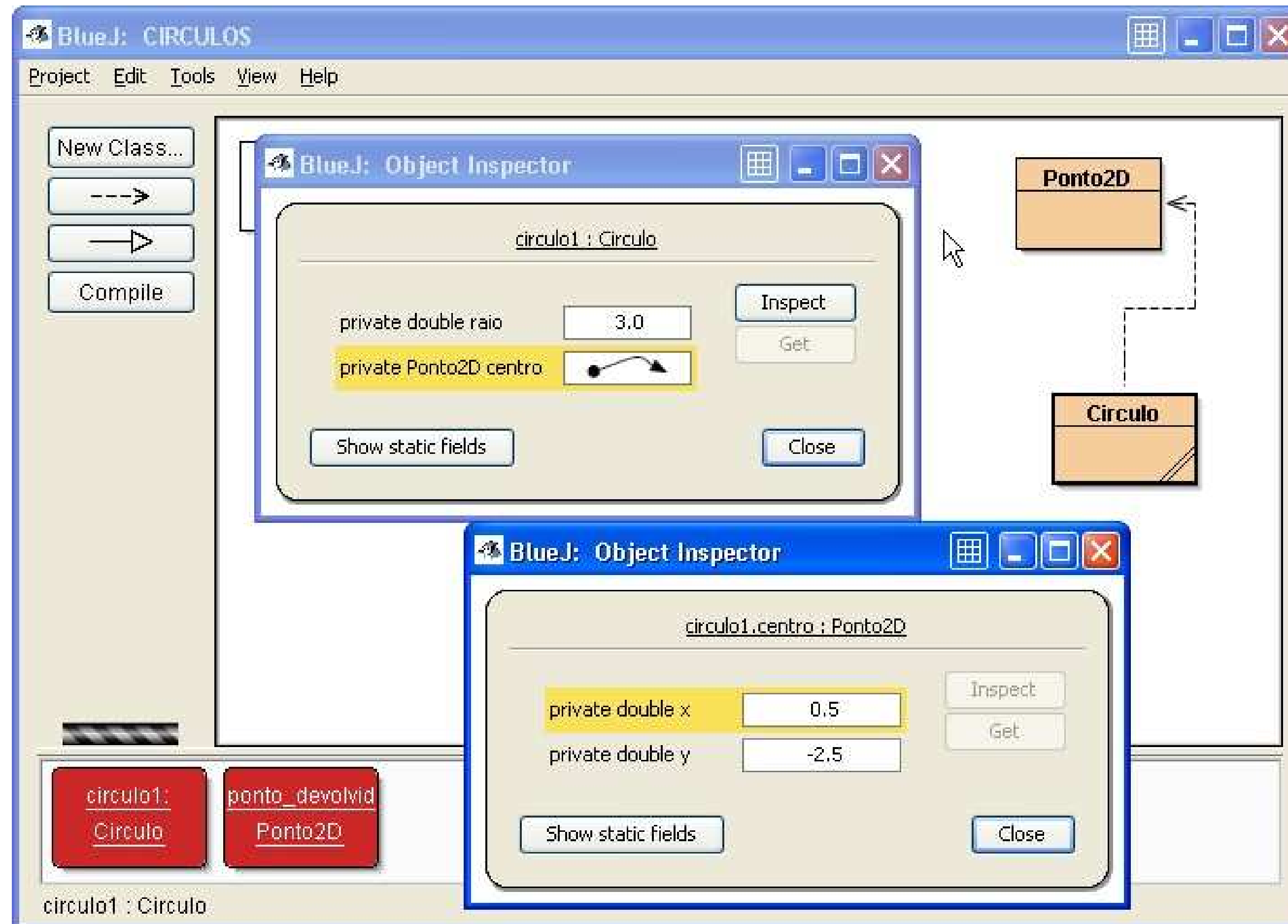
© F. Mário Martins 2007

ALTERAÇÃO DO PONTO DEVOLVIDO

INSPECT DO PONTO MODIFICADO



Porém, o Centro do Círculo (que estava partilhado) foi alterado também (e isto é muito mau !!)



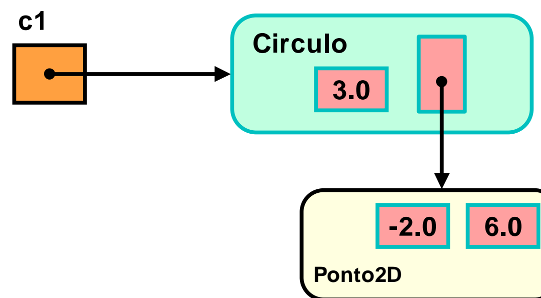
O PROBLEMA DA CLONAGEM DE OBJECTOS

- **"shallow" clone (cópia parcial que deixa endereços partilhados)**
- **"deep" clone (cópia em que nenhum objecto partilha endereço com outro)**

EXEMPLO: CLONAGEM DE UM CIRCULO

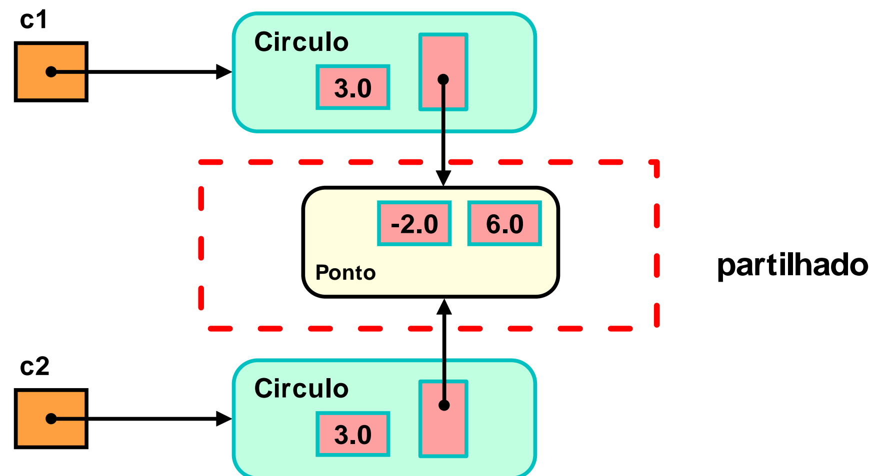
CRIAÇÃO DE UM CIRCULO USANDO O CONSTRUTOR `Circulo(Ponto2D p, double r)`

```
c1 = new Circulo( new Ponto2D(-2.0, 6.0), 3.0 );
```



CÓPIA "SHALLOW"

```
public Circulo clone1() { return new Circulo(centro, raio); }

   c2 = c1.clone1();
```



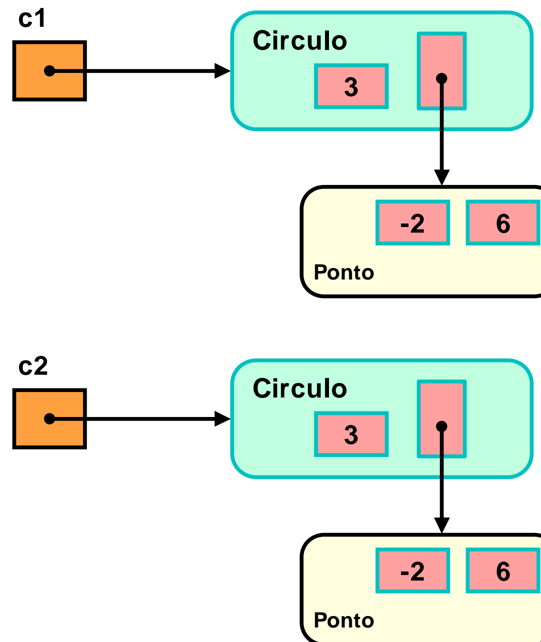
- "SHALLOW CLONE" – HÁ PARTILHA DE "PARTE" DO OBJECTO CLONADO; IGUAL A PARTILHAR TODO !!
- ALTERAÇÕES NO CENTRO DE c2 ALTERAM O CENTRO DE c1.

© F. Mário Martins 2007

CÓPIA “DEEP”

```
public Circulo clone2() { return new Circulo(centro.clone(), raio); }
```

```
c2 = c1.clone2();
```



“DEEP CLONE” – NÃO HÁ PARTILHA DE QUALQUER “PARTE” DO OBJECTO CLONADO

REGRA: CLONE DO TODO = CLONE DAS PARTES, SENDO QUE TIPOS SIMPLES E OBJECTOS IMUTÁVEIS (CF. STRING, INTEGER, FLOAT, ...) NÃO NECESSITAM DE SER “CLONADOS” (SÃO VALORES).

© F. Mário Martins 2007

ADMITINDO QUE NA CLASSE PONTO2D TEMOS DEFINIDO O CONSTRUTOR

```
public Ponto2D(Ponto2D pp) {
    x = pp.getX(); y = pp.getY();
}
```

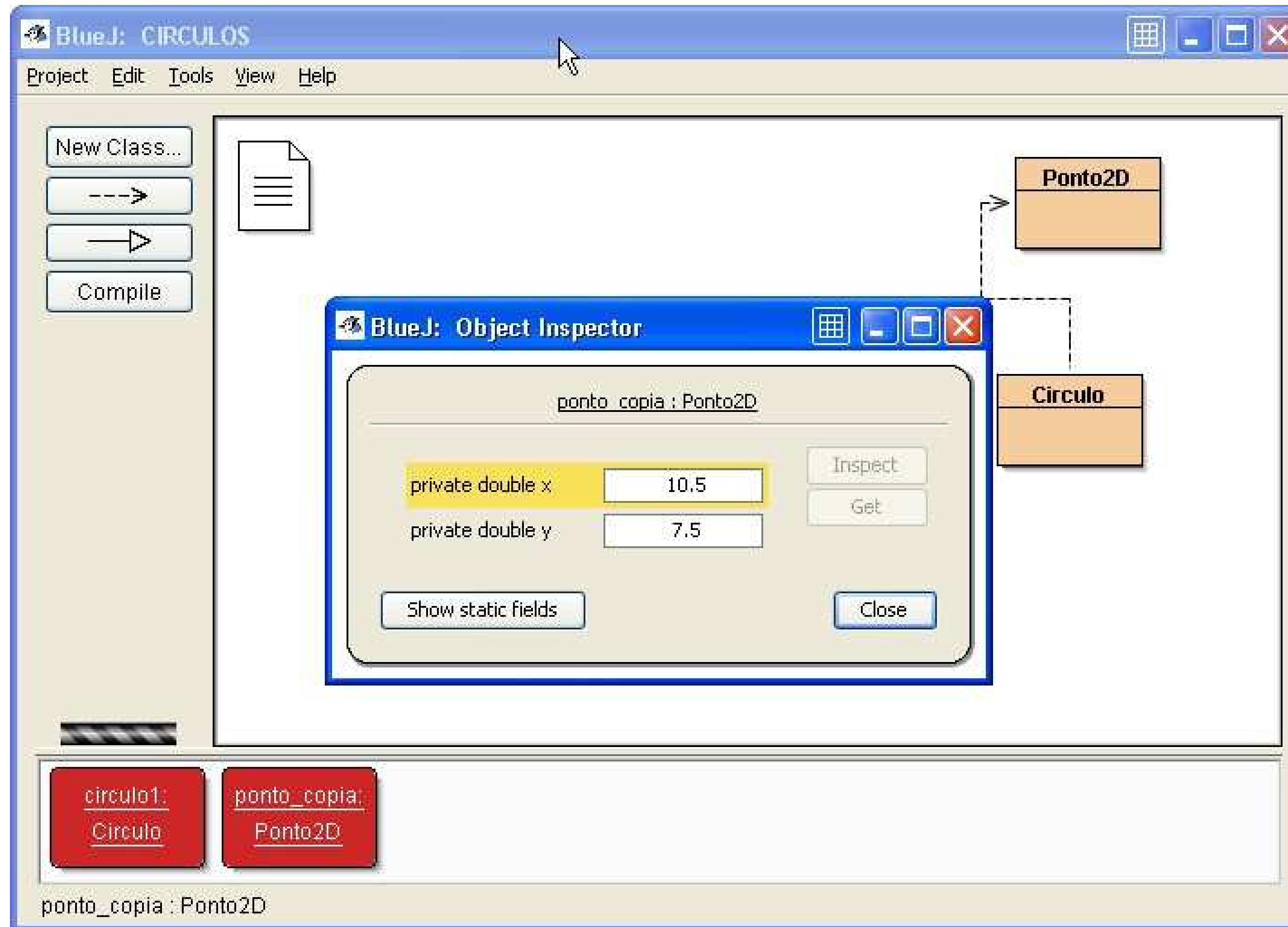
E AINDA UM MÉTODO DE CÓPIA DE PONTOS DEFINIDO COMO:

```
public Ponto2D clone() {
    return new Ponto2D(this);
}
```

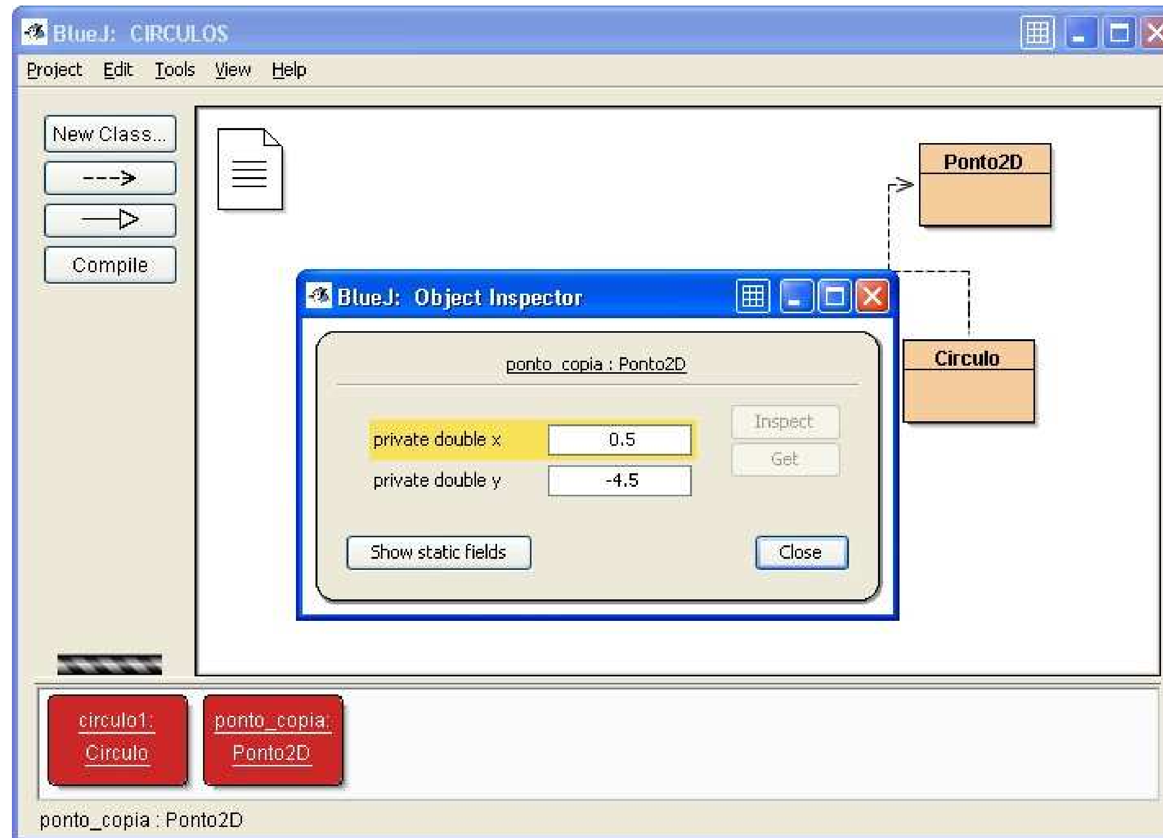
TESTEMOS AGORA COM O MÉTODO GETCENTRO() DEFINIDO COMO:

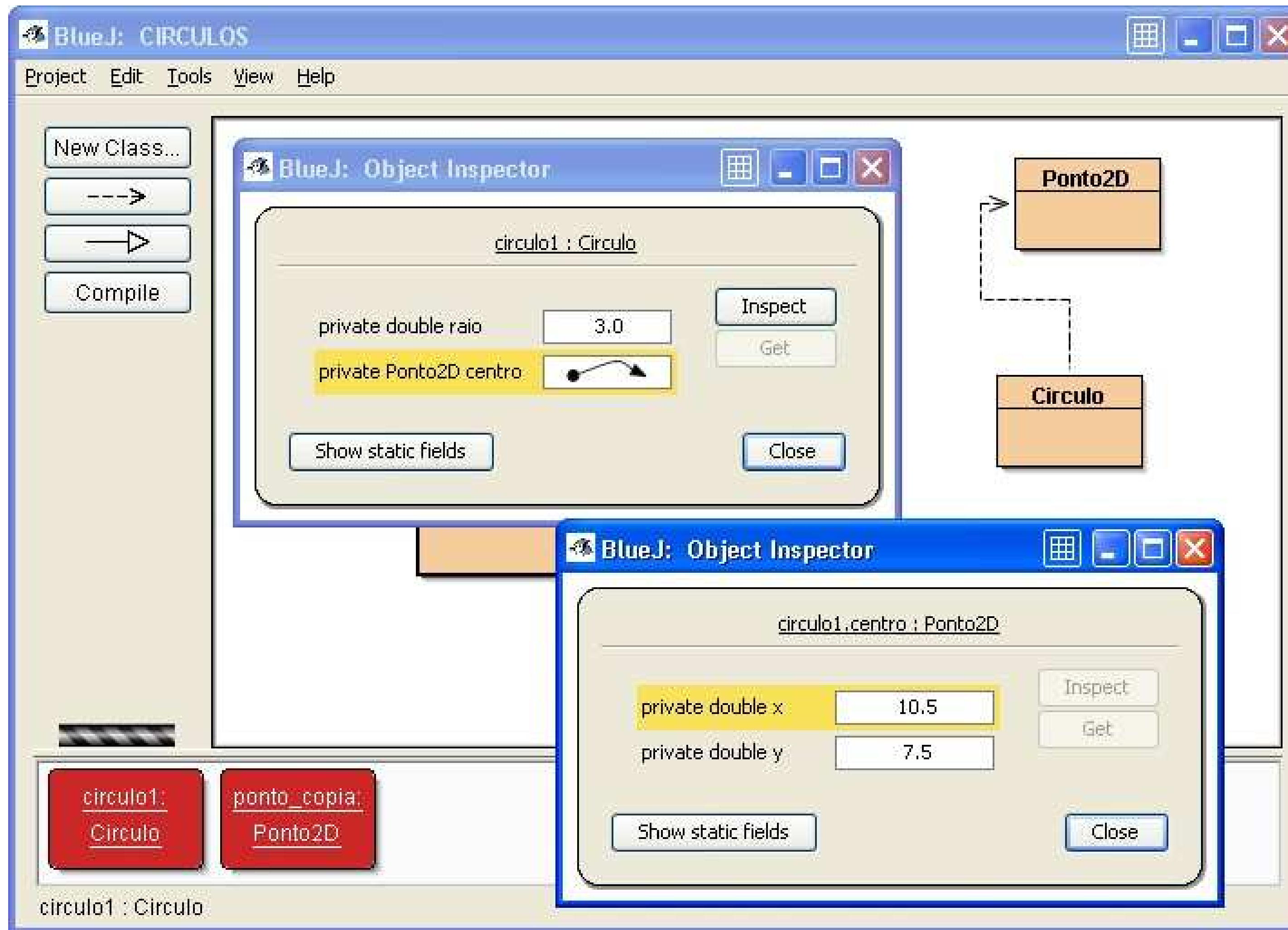
```
/**
 * Devolve uma CÓPIA do ponto que representa o CENTRO. O que for exteriormente
 * feito com esta CÓPIA NÃO MODIFICA o CENTRO do círculo pois
 * não têm endereços partilhados.
 */

public Ponto2D getCentro() {
    return centro.clone(); // cria um novo Ponto2D, cópia do centro !!
}
```

VEJAMOS SE COPIOU BEM ...



VAMOS ALTERAR ESTE PONTO E VER SE AGORA O CENTRO É MUDADO ...





AGORA ESTÁ TUDO OK. NÃO HÁ PARTILHA DE ENDEREÇOS.
O ENCAPSULAMENTO É PRESERVADO.

REGRA: DEVOLVER CLONES DAS VARIÁVEIS DE INSTÂNCIA DE TIPO REFERENCIADO

ENCAPSULAMENTO vs. PARTILHA DE ENDEREÇOS

TEOREMA2:

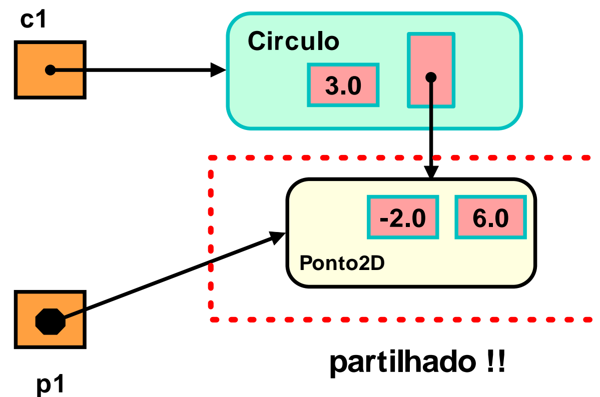
QUANDO UM MÉTODO DE INSTÂNCIA OU CONSTRUTOR RECEBE COMO PARÂMETRO UM OBJECTO E O ATRIBUI DIRECTAMENTE A UMA VARIÁVEL DE INSTÂNCIA, A VARIÁVEL DE INSTÂNCIA E O OBJECTO EXTERNO TÊM O MESMO ENDEREÇO, PELO QUE SE NO EXTERIOR ALGUÉM MODIFICAR O VALOR DE TAL OBJECTO O VALOR DA VARIÁVEL DE INSTÂNCIA É TAMBÉM MODIFICADO.

EXEMPLO – CLASSE CIRCULO:

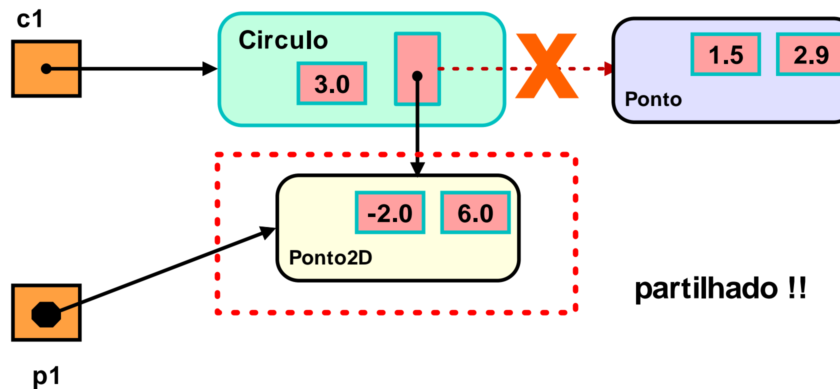
```
import static java.lang.Math.PI;
public class Circulo {

   private double raio;     // o raio do círculo
   private Ponto2D centro;  // ponto que define o centro do círculo

   // Construtores de círculos
   public Circulo(Ponto2D pcentro, double praio) {   // ponto parâmetro directamente atribuído
       raio = praio; centro = pcentro;
   }

…….
   // Métodos de Instância
……………
   public void mudaCentro(Ponto2D nc) { centro = nc; }   // ponto parâmetro directamente atribuído


}
```

CONFIGURAÇÕES: EXEMPLOS

```
public Circulo(Ponto2D pcentro, double praio) { centro = pcentro; raio = praio; }
Circulo c1 = new Circulo(p1, 3.0);
```




```
Circulo c1 = new Circulo( new Ponto2D(1.5, 2.9), 3.0) ;
public void mudaCentro(Ponto2D nc) { centro = nc; }
 c1.mudaCentro(p1);
```

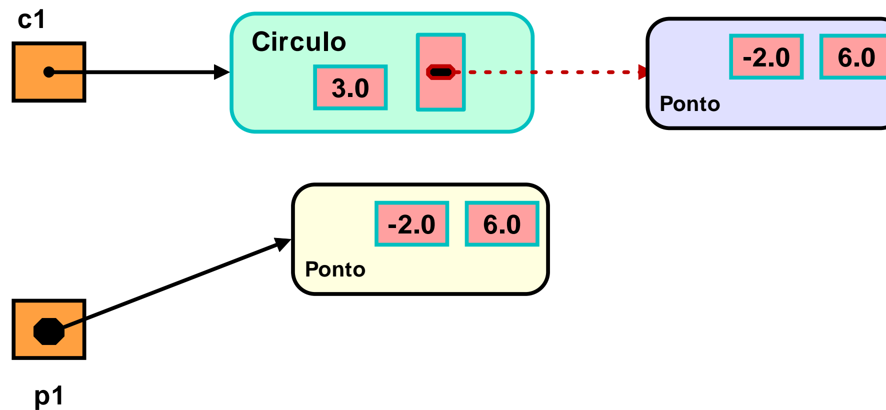


© F. Mário Martins 2007

CONSTRUTORES CORRECTOS (FAZEM CLONE() DO PARÂMETRO)

```
public Circulo(Circulo c) { centro = c.getCentro().clone(); raio = c. getRaio(); }
public Circulo(Ponto2D pcentro, double praio) { centro = pcentro.clone(); raio = praio; }
```

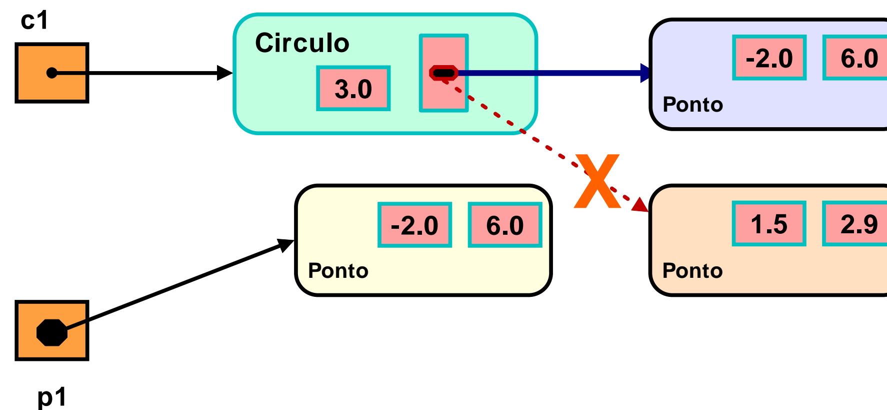
```
Circulo c1 = new Circulo(p1, 3.0);
```



NÃO HÁ PARTILHA DE OBJECTOS ENTRE c1 E p1 !!

MODIFICADORES CORRECTOS (FAZEM CLONE() DO PARÂMETRO)

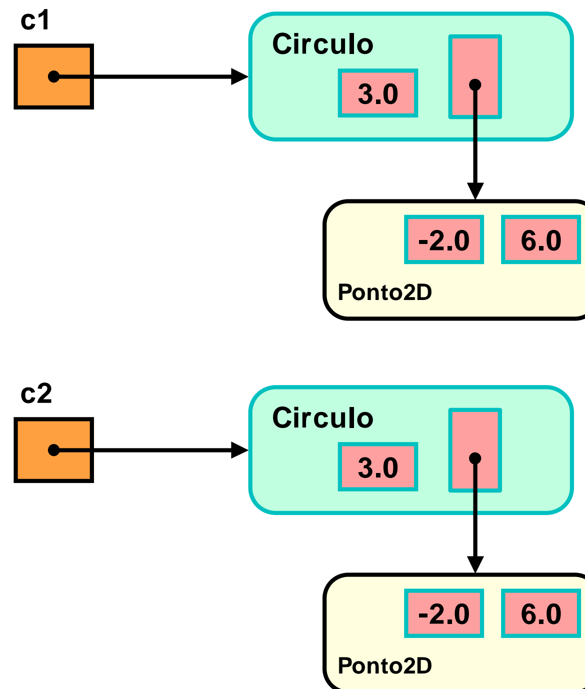
```
Circulo c1 = new Circulo( new Ponto2D(1.5, 2.9), 3.0) ;

public void mudaCentro(Ponto2D nc) { centro = nc.clone(); }
 c1.mudaCentro(p1);
```



NÃO HÁ PARTILHA DE OBJECTOS ENTRE `c1` E `p1` !!

CLONE NORMALIZADO – USA CONSTRUTOR DE CÓPIA

```
public Circulo clone() { return new Circulo(this); }

c2 = c1.clone();
```



TRATA-SE SEMPRE DE UMA "DEEP COPY" PORQUE NÃO HÁ ENDEREÇOS PARTILHADOS ENTRE O OBJECTO ORIGINAL E A CÓPIA PRODUZIDA, TAL COMO SE PRETENDE.

© F. Mário Martins 2007

MÉTODO clone( ) DE CLASSES JAVA

- Objectos de tipo String e das Wrapper Classes (cf. Integer, Float, Double) não necessitam de ser clonados porque são imutáveis;

- O Método clone() definido nas classes de Java é sempre "shallow" (é uma cópia com partilha) e devolve sempre uma instância da classe Object; Assim, em seguida devemos fazer o "casting" para o tipo correcto.

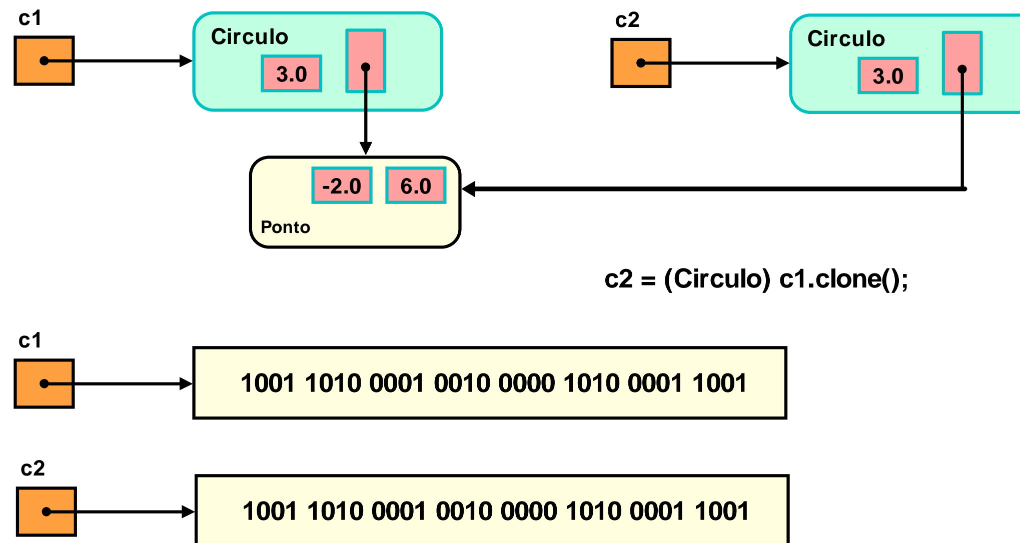
```
GregorianCalendar nascidoEm, copiaShallow;
nascidoEm = new GregorianCalendar(1990, 3, 21, 0, 0);
copiaShallow = (GregorianCalendar) nascidoEm.clone();
```

→ Object

```
CanvasPane canvas = new CanvasPane();
……
canvasImage = canvas.createImage(size.width, size.height);
graphic.fillRect(0, 0, size.width, size.height);
……
graphic1 = (Graphics2D) graphic.clone();
```

- O Método `clone()` genérico de Java, definido na classe Object é sempre "shallow" porque apenas copia o array de bits que representa o objecto sem saber o que tais bits representam (apontadores ou valores).

  Poderia ser usado em classes nossas mas o resultado não é o que se pretende e ainda teríamos que fazer "casting" !!



REGRA: Definir sempre os nossos métodos "deep" clone nas nossas classes.
Esta regra será estendida às Colecções de Java a estudar a seguir.