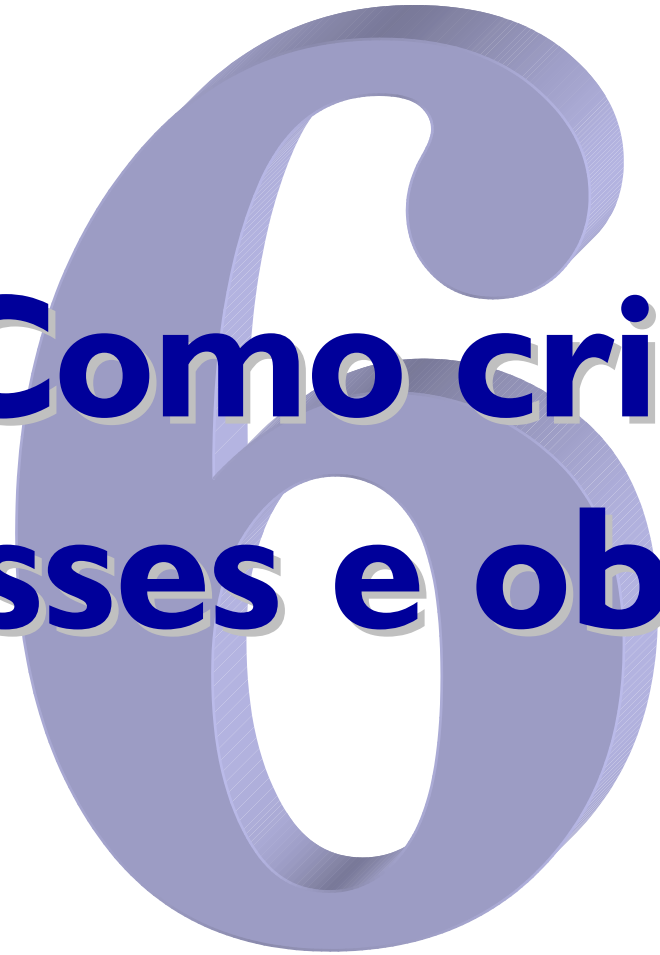


6 Como criar classes e objetos

Helder da Rocha
www.argonavis.com.br

Assuntos abordados

- Este módulo explora detalhes da construção de classes e objetos
  - Construtores
  - Implicações da herança
  - Palavras super e this, usadas como referências para o objeto corrente e a super classe
  - Instruções super() e this() usadas para chamar construtores durante a criação de objetos
  - Detalhes sobre a inicialização de objetos e possíveis problemas

Criação e destruição de objetos

- Para a criação de novos objetos, Java garante que cada classe tenha um construtor
  - O construtor default recebe zero argumentos
  - Faz apenas inicialização da superclasse
- Programador pode criar um construtor explicitamente e determinar suas operações de inicialização
  - Inicialização pela superclasse continua garantida
  - Construtor default deixa de existir
- Objetos são destruídos automaticamente pelo sistema, porém, sistema não faz finalização
  - Método finalize(), herdado de Object, teoricamente permite ao programador controlar a finalização de qualquer objeto
  - finalize() não funciona 95% das vezes - não use! Se precisar de finalização, coloque seu código em um bloco try {...} finally {...}

Construtores e sobrecarga

- *Construtores default (sem argumentos) só existem quando não há construtores definidos explicitamente no código*
  - *A criação de um construtor explícito substitui o construtor fornecido implicitamente*
- *Uma classe pode ter vários construtores (isto se chama sobrecarga de nomes)*
  - *Distinção é feita pelo número e tipo de argumentos (ou seja, pela assinatura do construtor)*
- *A assinatura é a identidade do método. É pela assinatura que ele se distingue dos outros métodos. Consiste de*
  - *Tipo de retorno*
  - *Nome*
  - *Tipo de argumentos*
  - *Quantidade de argumentos*

Sobrecarga de métodos

- Uma classe também pode ter vários métodos com o mesmo nome (sobrecarga de nomes de métodos)
  - Distinção é feita pela assinatura: tipo e número de argumentos, assim como construtores
  - Apesar de fazer parte da assinatura, o tipo de retorno não pode ser usado para distinguir métodos sobrecarregados
- Na chamada de um método, seus parâmetros são passados da mesma forma que em uma atribuição
  - Valores são passados em tipos primitivos
  - Referências são passadas em objetos
  - Há promoção de tipos de acordo com as regras de conversão de primitivos e objetos
  - Em casos onde a conversão direta não é permitida, é preciso usar operadores de coerção (cast)

Distinção de métodos na sobrecarga

- *Métodos sobrecarregados devem ser diferentes o suficiente para evitar ambigüidade na chamada*
- *Qual dos métodos abaixo ...*

```
int metodo (long x, int y) {...}
int metodo (int x, long y) {...}
```

- *... será chamado pela instrução abaixo?*

```
int z = metodo (5, 6);
```

- *O compilador detecta essas situações*

this()

- *Em classes com múltiplos construtores, que realizam tarefas semelhantes, this() pode ser usado para chamar outro construtor local, identificado pela sua assinatura (número e tipo de argumentos)*

```
public class Livro {
   private String titulo;
   public Livro() {
      titulo = "Sem titulo";
   }

   public Livro(String titulo) {
      this.titulo = titulo;
   }
}
```

```
public class Livro {
   private String titulo;
   public Livro() {
      this("Sem titulo");
   }

   public Livro(String titulo) {
      this.titulo = titulo;
   }
}
```

super()

- Todo construtor chama algum construtor de sua superclasse
  - Por default, chama-se o construtor sem argumentos, através do comando super() (implícito)
  - Pode-se chamar outro construtor, identificando-o através dos seus argumentos (número e tipo) na instrução super()
  - super(), se presente, deve sempre ser a primeira instrução do construtor (substitui o super() implícito)
- Se a classe tiver um construtor explícito, com argumentos, subclasses precisam chamá-lo diretamente
  - Não existe mais construtor default na classe

this e super

- A palavra this é usada para referenciar membros de um objeto
  - Não pode ser usada dentro de blocos estáticos (não existe objeto atual 'this' em métodos estáticos)
  - É obrigatória quando há ambiguidade entre variáveis locais e variáveis de instância
- super é usada para referenciar os valores originais de variáveis ou as implementações originais de métodos sobrepostos

```
class Numero {
    public int x = 10;
}
```

```
class OutroNumero extends Numero {
    public int x = 20;
    public int total() {
        return this.x + super.x;
    }
}
```

20

10

- Não confunda this e super com this() e super()
  - Os últimos são usados apenas em construtores!

Inicialização de instâncias

- O que acontece quando um objeto é criado usando *new NomeDaClasse()* ?
  - 1. Inicialização default de campos de dados (0, null, false)
  - 2. Chamada recursiva ao construtor da superclasse (até Object)

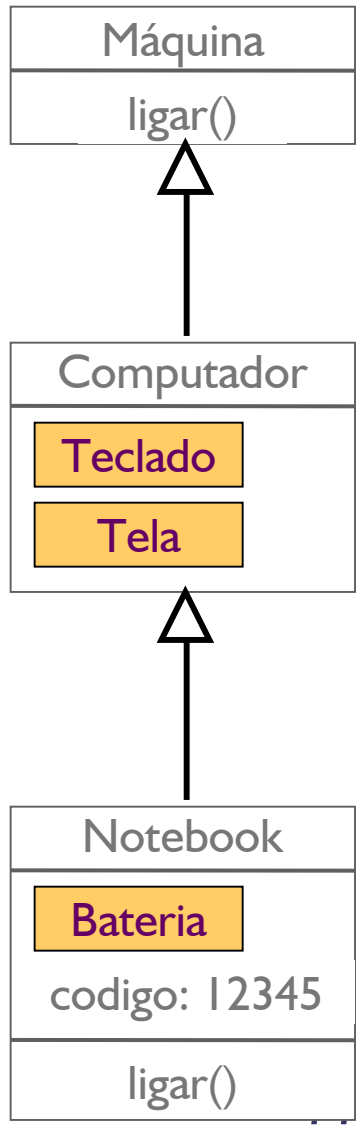
    2.1 Inicialização default dos campos de dados da superclasse (recursivo, subindo a hierarquia)

    2.2 Inicialização explicita dos campos de dados

    2.3 Execução do conteúdo do construtor (a partir de Object, descendo a hierarquia)
  - 3. Inicialização explícita dos campos de dados
  - 4. Execução do conteúdo do construtor

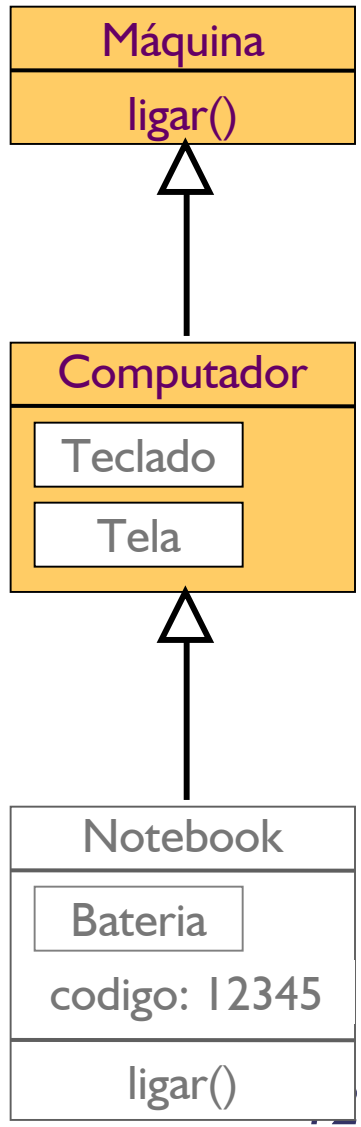
Exemplo (1)

```
class Bateria {
  public Bateria() {
    System.out.println("Bateria()");
  }
}

class Tela {
  public Tela() {
    System.out.println("Tela()");
  }
}

class Teclado {
  public Teclado() {
    System.out.println("Teclado()");
  }
}
```



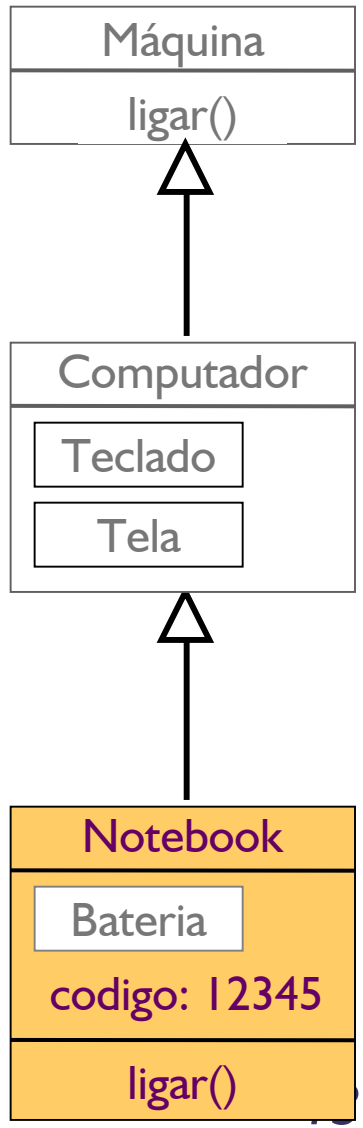
Exemplo (2)

```
class Maquina {
  public Maquina() {
     System.out.println("Maquina()");
     this.ligar();
  }
  public void ligar() {
    System.out.println("Maquina.ligar()");
  }
}

class Computador extends Maquina {
  public Tela tela = new Tela();
  public Teclado teclado = new Teclado();
  public Computador() {
     System.out.println("Computador()");
  }
}
```

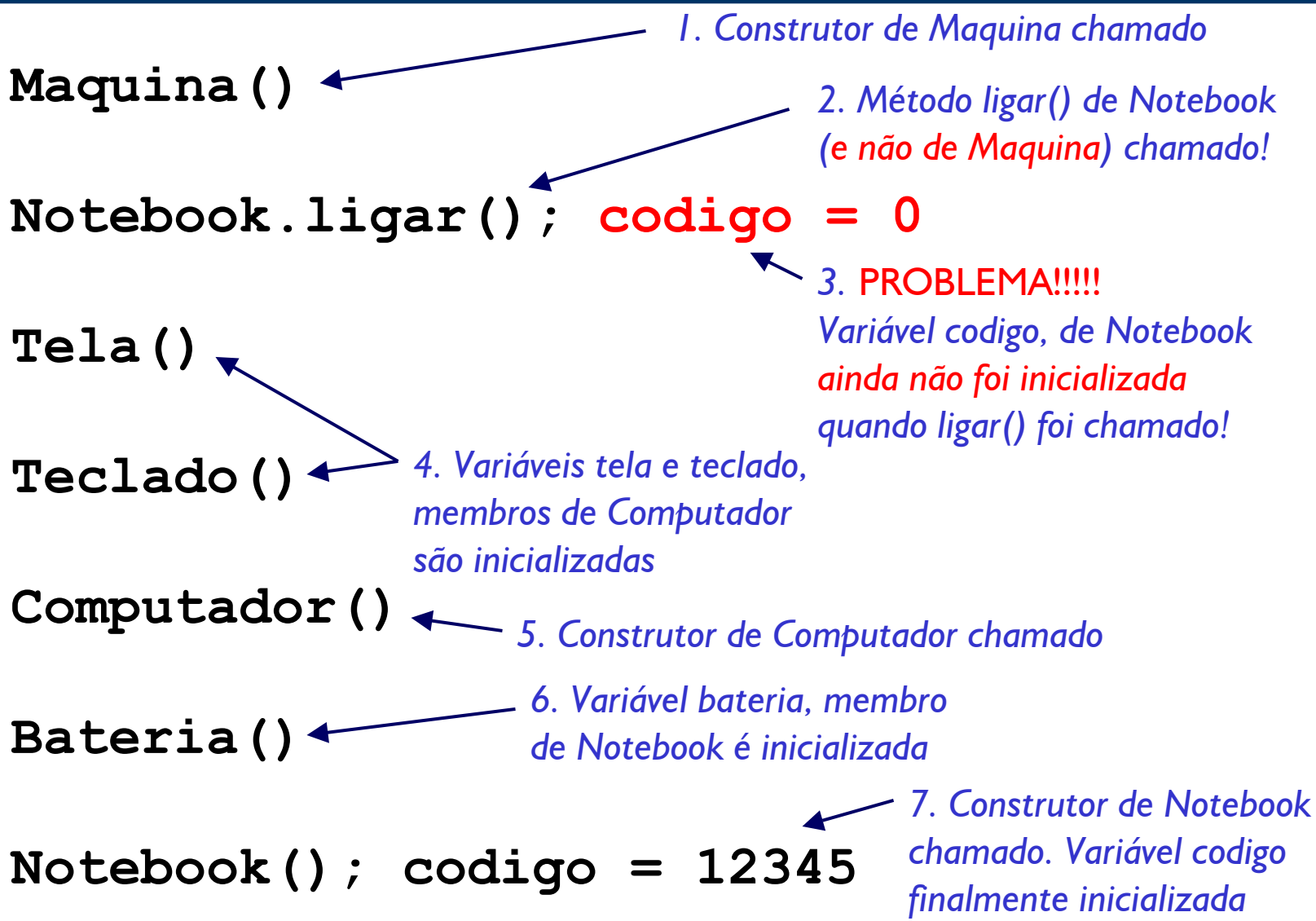


Exemplo (3)

```
class Notebook extends Computador {
  int codigo = 12345;
  public Bateria bateria = new Bateria();
  public Notebook() {
    System.out.print("Notebook(); " +
                     "codigo = "+codigo);
  }
  public void ligar() {
    System.out.println("Notebook.ligar();" +
                       " codigo = "+ codigo);
  }
}

public class Run {
  public static void main (String[] args) {
  new Notebook();
  }
}
```



Resultado de `new Notebook()`



Detalhes

N1. new Notebook() chamado
N2. variável código iniciaizada: 0
N3. variável bateria iniciaizada: null
N4. super() chamado (Computador)

C1. variável teclado iniciaizada: null
C2. variável tela iniclaizada: null
C3. super() chamado (Maquina)

M2. super() chamado (Object)

M2. Corpo de Maquina() executado:
println() e this.ligar()

C4: Construtor de Teclado chamado

Tk1: super() chamado (Object)

C5. referência teclado inicializada
C6: Construtor de Tela chamado

Te1: super() chamado (Object)

C7: referência tela inicializada
C8: Corpo de Computador()
executado: println()

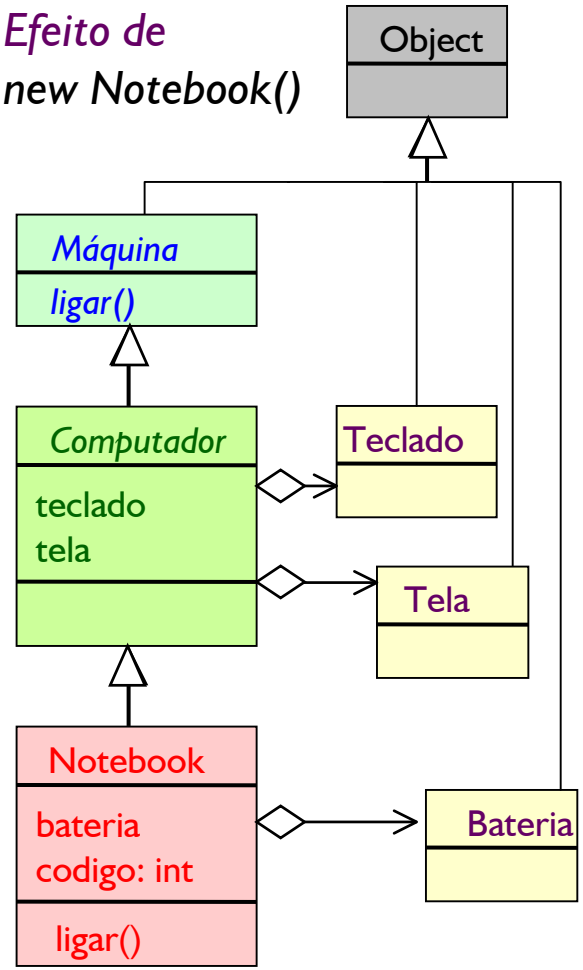
N5. Construtor de Bateria chamado

B1: super() chamado (Object)

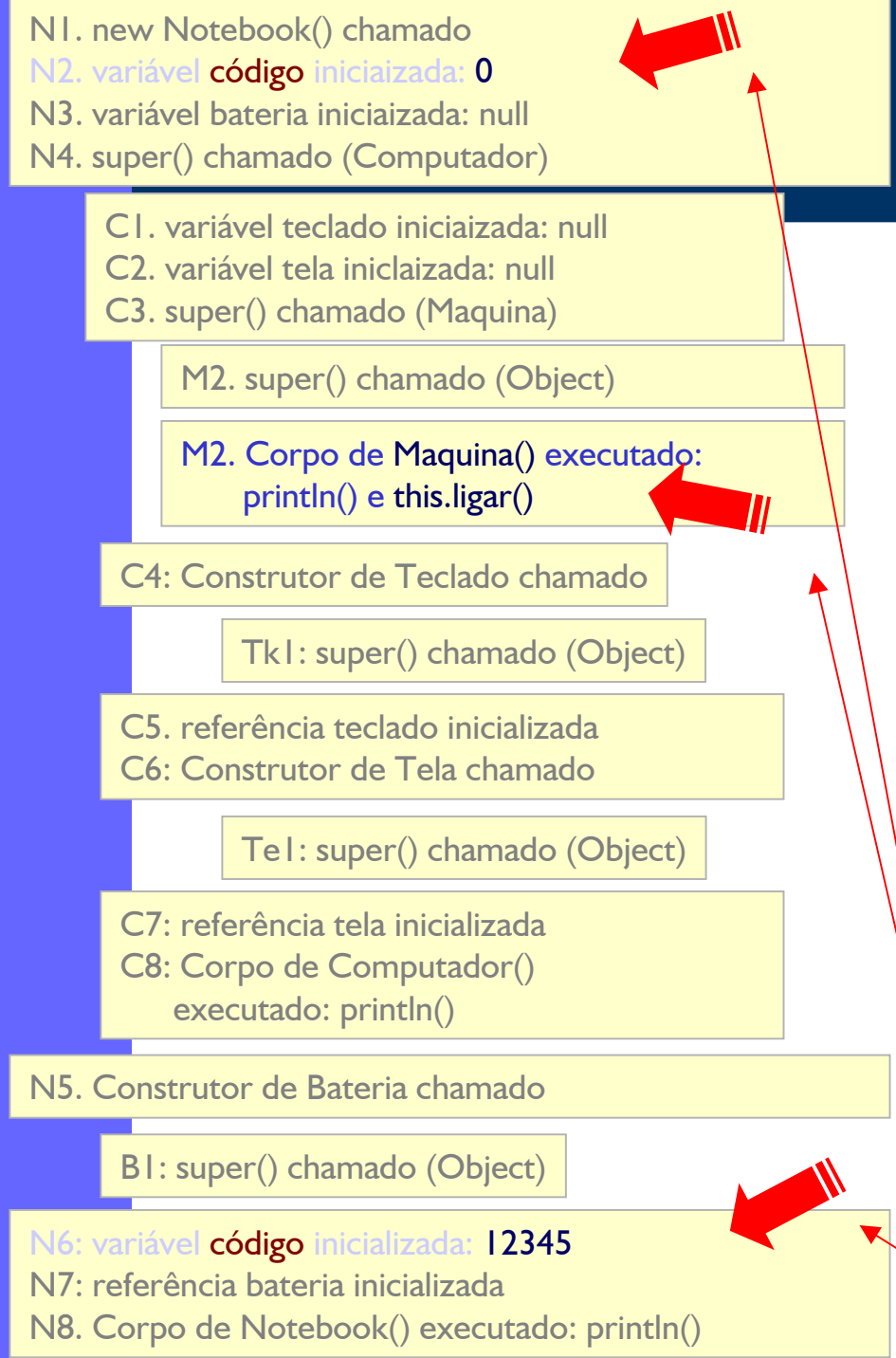
N6: variável código inicializada: 12345
N7: referência bateria inicializada
N8. Corpo de Notebook() executado: println()

O1. Campos inicializados
O2. Corpo de Object() executado

Efeito de
new Notebook()



Problemas com inicialização



- *método ligar() é chamado no construtor de Maquina, mas ...*
- *... a versão usada é a implementação em Notebook, que imprime o valor de código (e não a versão de Maquina como aparenta)*
- *Como código ainda não foi inicializado, valor impresso é 0!*

Preste atenção nos pontos críticos!

Como evitar o problema?

- Evite chamar métodos locais dentro de construtores
  - Construtor (qualquer um da hierarquia) sempre usa versão sobreposta do método
- Isto pode trazer resultados inesperados se alguém estender a sua classe com uma nova implementação do método que
  - Dependa de variáveis da classe estendida
  - Chame métodos em objetos que ainda serão criados (provoca NullPointerException)
  - Dependa de outros métodos sobrepostos
- Use apenas métodos finais em construtores
  - Métodos declarados com modificador final não podem ser sobrepostos em subclasses

Inicialização estática

- *Para inicializar valores estáticos, é preciso atuar logo após a carga da classe*
  - *O bloco 'static' tem essa finalidade*
  - *Pode estar em qualquer lugar da classe, mas será chamado antes de qualquer outro método ou variável*

```
class UmaClasse {
    private static Point[] p = new Point[10];
    static {
       for (int i = 0; i < 10; i++) {
          p[i] = new Point(i, i);
       }
    }
}
```

- *Não é possível prever em que ordem os blocos static serão executados, portanto: só tenha um!*

Exercício

- 1. Preparação
  - a) Crie um novo projeto para este módulo
  - b) Copie os arquivos Ponto e Circulo dos exercícios feitos no Capítulo 2 (se você não os fez, use os da solução)
- 2. Crie mais um construtor em Círculo
  - a) Crie um construtor default, que represente um círculo na origem (0,0) com raio unitário
  - b) Use this() em dois dos três construtores de Circulo
- 3. Crie uma classe Ponto3D que estenda Ponto.
  - Use, se necessário, a chamada correta de super() para permitir que a classe seja compilada
- 4. Altere TestaCirculo para testar as novas classes

Curso J100: Java 2 Standard Edition

Revisão 17.0

(helder@acm.org)

argonavis.com.br