

## Como criar classes e objetos, métodos e variáveis

*Helder da Rocha*  
[www.argonavis.com.br](http://www.argonavis.com.br)

# 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*

# 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 (sobrecarga de nomes)*
  - *distinção é feita pelo número e tipo de argumentos (assinatura do construtor)*

# Sobrecarga de métodos

- *Uma classe também pode ter vários métodos com o mesmo nome (sobrecarga)*
  - *distinção é feita pelo tipo e número de argumentos, assim como construtores*
  - *o tipo de retorno não pode ser usado para distinguir métodos sobrecarregados*

# Promoção na sobrecarga

- 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)

```
metodo ( (byte) 10, (short) 15 );
```

# Distinção de métodos

- 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);
```

- *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)*

# 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;  
    }  
}
```

The diagram illustrates the relationship between the two classes. A black arrow points from the `OutroNumero` class box to the `Numero` class box, indicating inheritance. Inside the `OutroNumero` box, a grey arrow points from the value `20` (the local variable `x`) to the `this.x` part of the expression `this.x + super.x`. Another grey arrow points from the value `10` (the parent class variable `x`) to the `super.x` part of the same expression.



- *Em classes com múltiplos construtores, que realizam tarefas semelhantes, this() pode ser usado para chamar outro construtor local, identificado pelo 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;  
    }  
}
```

- Antes de um objeto ser destruído, pelo garbage collector, seu método `finalize()` (herdado de `Object`) é chamado
  - *programador pode sobrepor `finalize()` para realizar alguma finalização necessária ao objeto*

```
class Objeto {  
    ...  
    public void finalize() {  
        System.out.println("Objeto sendo destruído");  
        super.finalize();  
    }  
}
```

- O ideal, porém, é criar seu próprio método de finalização e chamá-lo diretamente. A Java Language Specification **não garante** que `finalize()` seja chamado. O seu funcionamento é dependente do JVM

# Inicialização de instâncias

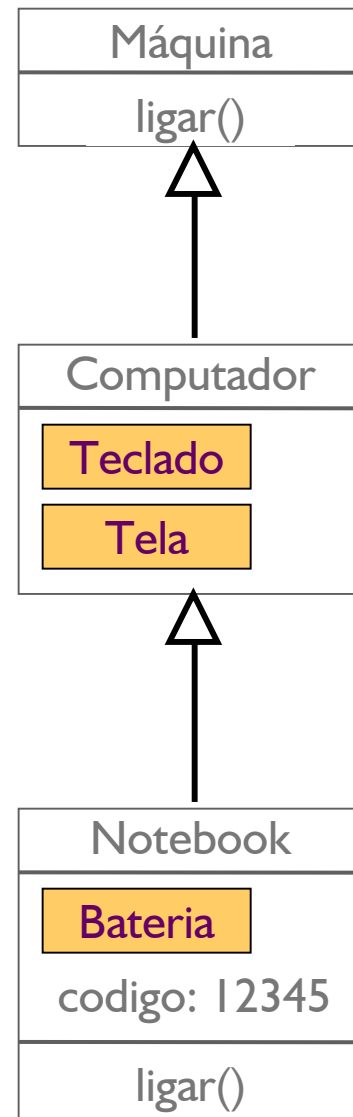
- O que acontece quando um objeto é criado usando *new NomeDaClasse()* ?
  - 1. inicialização default de campos de dados (0, null, false)
  - 2. chamada recursiva ao construtor da superclasse (até Object)
    - 2.1 inicialização default dos campos de dados da superclasse (recursivo, subindo a hierarquia)
    - 2.2 inicialização explícita 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()");  
    }  
}
```

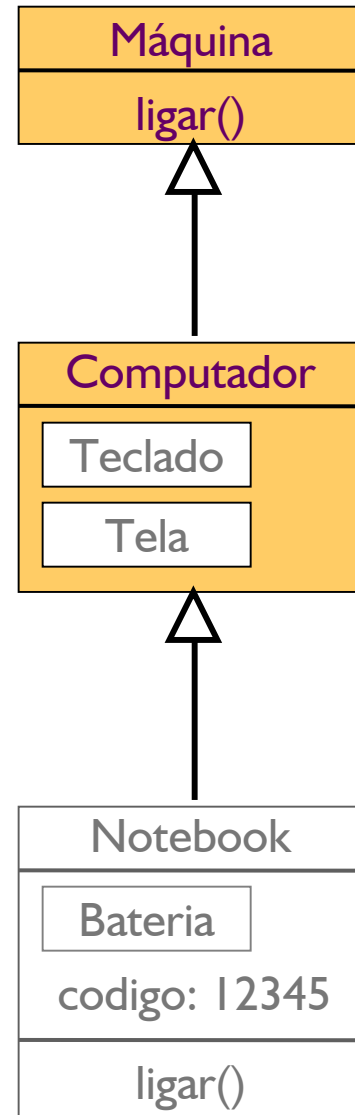
```
class Teclado {  
    public Teclado() {  
        System.out.println("Teclado()");  
    }  
}
```



# Exemplo (2)

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

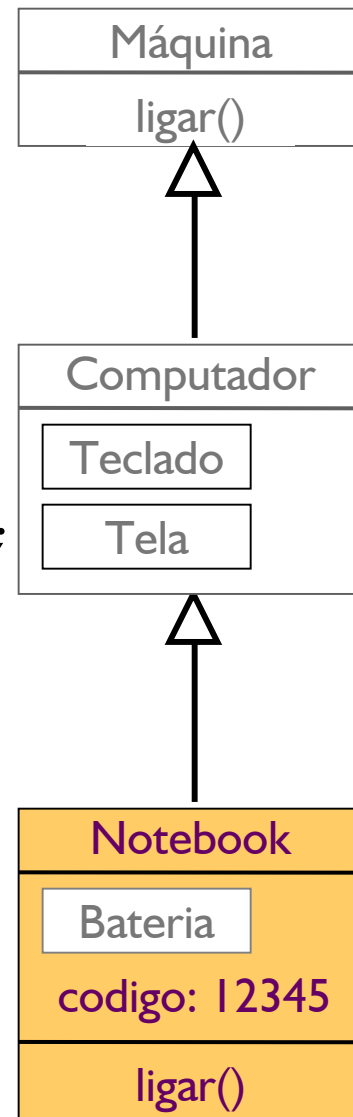
```
class Computador extends Maquina {  
    public Tela tela = new Tela();  
    public Teclado teclado = new Teclado();  
    public Computador() {  
        System.out.println("Computador()");  
    }  
}
```



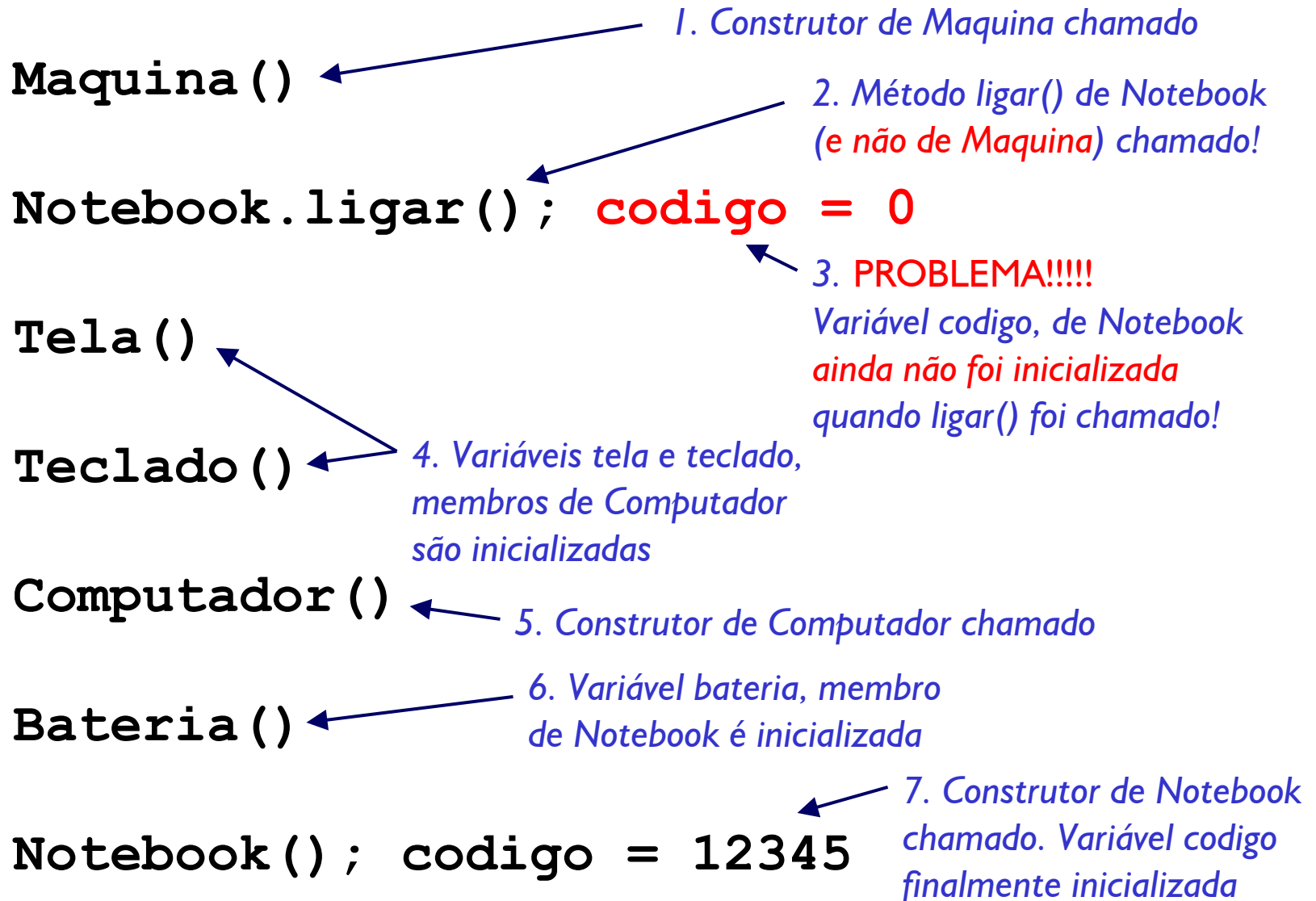
# Exemplo (3)

```
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 inicializada: 0**  
**N3. variável bateria inicializada: null**  
**N4. super() chamado (Computador)**

**C1. variável teclado inicializada: null**  
**C2. variável tela inicializada: 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)**

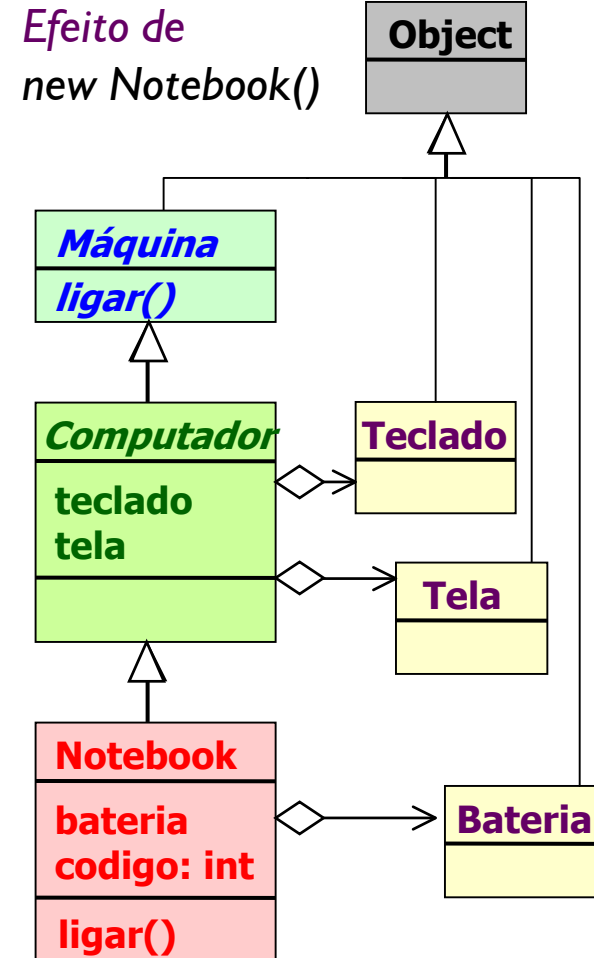
**C7: referência tela inicializada**  
**C8: Corpo de Computador() executado: println()**

**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**





# Problemas com inicialização

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

C1. variável teclado inicializada: null  
C2. variável tela inicializada: 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()

- 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*

# 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);  
        }  
    }  
}
```