

# PROGRAMAÇÃO MODULAR

## FUNDAMENTOS DE POO: CONSTRUTORES E DESTRUTORES

PROF. JOÃO CARAM

# CONSTRUTORES

2

- Construtores são usados para criar e iniciar objetos com valores diferentes do padrão:
  - Em geral, possuem o mesmo nome da classe.
  - Não possuem valores de retorno.
- Uma classe pode ter de um (oculto) a muitos construtores.

# BOAS RAZÕES PARA USAR CONSTRUTORES

3

- Estado inicial padrão pode não ser aceitável;
- Fornecer um estado inicial consistente é conveniente ao criar objetos;
- Construir um objeto aos poucos é desgastante;
- Um construtor pode ser privado, restringindo assim o uso de sua classe.

# NA CLASSE PRODUTO...

---

```
Produto(String desc, double pr, int qt){  
    if(desc.length()>=3)  
        descricao = desc;  
    if(preco>0)  
        preco = pr;  
    if(qt >=0)  
        quant = qt ;  
}  
Produto(){  
    descricao = "Novo Produto";  
    preco = 0.01;  
    quant = 0;  
}
```

# USANDO PRODUTO COM CONSTRUTORES

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Produto p = new Produto();  
        p.descricao = "Xulambs";  
        p.preco = 1.99;  
        p.quant = 200;  
        System.out.println("Produto: " + p.descricao);  
        System.out.println("Preço : " + p.preco);  
        System.out.println("Estoque : " + p.quant);  
        if(p.temEstoque())  
            System.out.println("Produto em estoque.");  
        else  
            System.out.println("Produto em falta.");  
    }  
}
```

# USANDO PRODUTO COM CONSTRUTORES

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Produto p = new Produto();  
        p.descricao = "Xulambs";  
        p.preco = 1.99;  
        p.quant = 200;  
        System.out.println("Produto: " + p.descricao);  
        System.out.println("Preço : " + p.preco);  
        System.out.println("Estoque : " + p.quant);  
        if(p.temEstoque())  
            System.out.println("Produto em estoque.");  
        else  
            System.out.println("Produto em falta.");  
    }  
}
```

# USANDO PRODUTO COM CONSTRUTORES

---

```
class Aplicacao {  
    public static void main(String args[]){  
        Produto p = new Produto("Xulams", 1.99, 200);  
        System.out.println("Produto : " + p.descrever());  
        if(p.temEstoque())  
            System.out.println("Produto em estoque.");  
        else  
            System.out.println("Produto em falta.");  
    }  
}
```

# CONSTRUTORES

8

- Para que criar vários construtores para uma classe?
  - *Conforto de uso.*
  - *Cenários de uso.*





# CONSTRUTORES

9

- Para que criar vários construtores para uma classe?
  - *Conforto de uso.*
  - *Cenários de uso.*
- Mas, pensando em qualidade e nos pecados no processo de desenvolvimento...

# CLASSE PRODUTO

```
Produto(String desc, double pr, int qt){  
    if(desc.length()>=3)  
        descricao = desc;  
    if(preco>0)  
        preco = pr;  
    if(qt >=0)  
        quant = qt ;  
}  
Produto(){  
    descricao = "Novo Produto";  
    preco = 0.01;  
    quant = 0;  
}
```

**Código  
duplicado!**

# MÉTODOS INICIALIZADORES

11

- Inicializam/validam a construção de um objeto.
  - Independência funcional;
  - Separação de responsabilidades.



# CLASSE PRODUTO

---

```
void init(String desc, double pr, int qt){  
    if(desc.length()>=3)  
        descricao = desc;  
    if(preco>0)  
        preco = pr;  
    if(qt >=0)  
        quant = qt ;  
}
```

# CLASSE PRODUTO

---

```
Produto(String desc, double pr, int qt){  
    init(desc, pr, qt);  
}
```

```
Produto(String desc, double pr){  
    init(desc, pr, 1);  
}
```

```
Produto(){  
    init("Novo produto", 0.01, 1);  
}
```

# DESTRUTORES

# COLETOR DE LIXO

15

- Processo que libera automaticamente memória que não está sendo mais utilizada.
- Eliminam a necessidade de se desalocar memória explicitamente;
- Eliminam o vazamento de memória;
- Eliminam referências pendentes (*dangling pointer*).

# JAVA E COLETOR DE LIXO

16

- ▣ Linguagem Java:
  - ▣ não permite acesso direto à memória;
  - ▣ Não possui operadores de liberação de memória;
    - C, C++: *free, delete*
  - ▣ Possui coletor de lixo (garbage collector).



# JAVA E COLETOR DE LIXO

17

- Um objeto é elegível para coleta de lixo quando:
  - não é mais acessado por nenhuma referência;
  - referencia um outro objeto que também o referencia, formando um ciclo único e isolado.
- O coletor de lixo é autônomo.
- Método [System.gc\(\)](#)

# DESTRUTORES (*DESTRUCTORS*)

18

- Métodos especiais invocados quando um objeto é finalizado (capturado pelo coletor de lixo).
  - Só há um destrutor por classe.
  - Um destrutor não tem parâmetros.
  - Não deve ser chamado diretamente.

# DESTRUTORES (*DESTRUCTORS*)


19

- Objetivo: Liberação de recursos usados pelo objeto
- Ex: memória  
arquivos  
conexões de rede, bancos de dados..

# DESTRUTORES (C++ E C#)

---

```
~nomeDaClasse(){  
    //seu código aqui  
}
```

A large, light green watermark logo for PUC Minas is positioned diagonally across the center of the slide. It features a shield with a stylized 'P' and 'M' inside, surrounded by various symbols like a cross, a gear, and a book. The text 'PUC Minas' is written in a large, bold, sans-serif font across the bottom of the logo.

# C# E DESTRUTORES

21

- Há uma classe estática GC em C#
  - *Chamada manual da coleta.*
- Classes em C# podem implementar a interface *IDisposable*
  - Método *Dispose()*
  - Liberação de recursos sem destruir o objeto

# DESTRUTORES (JAVA)

22

- São chamados automaticamente pelo coletor de lixo.
- Um destrutor *sobrescreve o método finalize()* da sua classe base.



# DESTRUTORES (JAVA)

23

MÉTODO *FINALIZE()*  
DA CLASSE BASE??



# DESTRUTORES (JAVA)

24

- A classe Object em Java tem um método finalize.
- Você pode escrever o finalize para sua classe.
  - Pode ser necessário resolver pendências antes de um objeto ser removido.



# MÉTODO *FINALIZE*

---

```
public class Aluno {  
    private static final File fonte = new File("alunos.txt");  
    private Scanner leitor = new Scanner(fonte);  
    ...  
    @Override  
    protected void finalize() throws Throwable {  
        super.finalize();  
        if(leitor!=null) {  
            leitor.close();  
            leitor = null;  
        }  
    }  
}
```

# OBRIGADO.

DÚVIDAS?