# Aula 7 – POO 1
# Construtores

Profa. Elaine Faria
UFU - 2021

# Sobre o Material

- Agradecimentos
  - Aos professores José Gustavo e Fabiano, por gentilmente terem cedido seus materiais.

- Os slides consistem de adaptações e modificações dos slides dos professores José Gustavo e Fabiano

# Introdução

- Até o momento, temos tratado a inicialização dos atributos das classes de três maneiras:
  - Alteração dos valores dos atributos diretamente - atributos públicos - ruim!
  - Criação de métodos do tipo "cadastra" - método pode ser chamado a qualquer momento
  - Métodos do tipo ***set*** - adequados para alterações
- A ideia é determinar os valores dos atributos no momento de criação dos objetos - valor inicial dos atributos

# Construtores

- Métodos especiais de uma classe, utilizados para **INICIALIZAR** parâmetros e **EXECUTAR PROCEDIMENTOS** de inicialização
- Distinguem-se de outros métodos por possuírem o mesmo nome da classe e não podem ter retorno
- Uma classe pode possuir mais de um construtor
- Toda classe vem com um construtor padrão, que não inicializa nenhum atributo e nem realiza nenhuma operação

# Construtores

- Se um construtor for criado para uma classe, seu construtor padrão será excluído, e ficarão valendo apenas os construtores criados
- Apesar de permitido a uma classe possuir mais de um construtor, eles não podem possuir os mesmos tipos de parâmetros na mesma ordem

# Construtores

- Construtores NÃO RETORNAM nenhum tipo de dado, nem *void*
- Se uma classe não incluir um construtor, os atributos dos objetos dessa classe serão inicializados com seus valores padrão
- É interessante que o programador forneça sempre um construtor para uma classe
  - Segurança de que todos os atributos sejam inicializados corretamente, de acordo com as regras de uma aplicação

# Exemplo 1 – Conta

```
public class Conta {

    private String nome;
    private int numero;
    private int senha;
    private float saldo;
    private String tipo;

    public Conta(int numeroCor) {
        numero = numeroCor;
        System.out.println("Conta Cadastrada.");
    }

    public Conta(String nomeCor) {
        nome = nomeCor;
        System.out.println("Conta Cadastrada.");
    }
```

# Exemplo 1 – Conta

- Perceba que a classe Conta possui dois construtores
  - Todos possuem o mesmo nome, que por sua vez é o mesmo nome da classe
  - Eles possuem parâmetros de tipos diferentes
  - O primeiro inicializa o número da conta, assim que é chamado
  - O segundo inicializa o nome do correntista, assim que é chamado

# Exemplo 1 – Conta

```
public class Conta {

    private String nome;
    private int numero;
    private int senha;
    private float saldo;
    private String tipo;

    public Conta(int numeroCor) {
        numero = numeroCor;
        System.out.println("Conta Cadastrada.");
    }

    public Conta(int senhaCor) {
        senha = senhaCor;
        System.out.println("Conta Cadastrada.");
    }
```

# Exemplo 1 – Conta

- O caso mostrado no slide anterior acusará um erro!

Por que?

# Exemplo 1 – Conta

- Os construtores são chamados para "construir" um objeto de uma determinada classe
  - Construir: alocar memória para o objeto sendo criado
- Dessa maneira, são chamados quando utilizamos o seguinte comando:

```
Conta c1 = new Conta();
           OU
Conta c1;
c1 = new Conta();
```

# Exemplo 1 – Conta

- Desta forma, para o exemplo apresentado, teremos

```
public static void main(String[] args) {

    Conta c1 = new Conta("Jose Gustavo");
    Conta c2 = new Conta(96016);

}
```

# Exercício 1

- Para a classe conta, crie um construtor para que, ao iniciar um objeto dessa classe, já sejam determinados o nome do correntista, seu número de conta, sua senha, e que o seu saldo seja de 500 reais

# Destrutores

- Métodos invocados para finalizar determinado objeto
  - Liberação de memória
  - Finalização de dispositivos ou subsistemas
- Java
  - Objetos descartados pelo Coletor de Lixo
  - Método ***finalize()*** torna um determinado objeto candidato a ser descartado
    - Utilizado para desalocar recursos adicionais
- C++
  - Método explicitamente criado pelo programador
  - Apenas um por classe

# Exemplo de Destrutor - C++

```
class vector {
    int sz; // número de elementos
    int* v; // ponteiro para inteiros
    public:
        vector(int); // construtor
        ~vector(); // destrutor
};
// Construtor
vector::vector (int s) {
    if (s<=0) {
       cout << "Bad Vector Size";
       return;
    }
    sz = s;
    v = new int[s]; // aloca um vetor de inteiros
    cout << "\n Construtor";
}
```

# Exemplo de Destrutor - C++

```
// Destrutor
vector::~vector() {
     delete [] v; // desaloca o vetor apontado por v
     cout << "\n Destrutor";
}



// Função Principal
void main() {
     vector v(10);
}
```

# Coleta de Lixo

- Java possui uma gerência automática de memória, isto é, quando um objeto não é mais referenciado pelo programa, ele é automaticamente coletado (destruído)
  - Esse processo se chama “coleta de lixo”
- Esgotamento de memória que acontecem em outras linguagens são menos prováveis no Java

# Coleta de Lixo

- Quando um objeto Java vai ser coletado, ele tem seu método **finalize** chamado
- Esse método deve efetuar qualquer procedimento de finalização que seja necessário antes da coleta do objeto

# Exemplo 2 – Máquina de Passagens

- Defina uma classe “Maquina de Passagens”, que represente uma máquina de emissão de passagens de ônibus, na qual o usuário insere dinheiro na máquina, e ela imprime e emite a passagem respectiva
- Considere que o preço do bilhete é R$2,00. Considere que a máquina aceite apenas notas de R$1,00 e de R$2,00. Considere também que os usuários não irão inserir outras notas
- A classe deverá ter os seguintes atributos:
  - Preço (representando o preço da passagem);
  - Quantidade Inserida (representando quanto o cliente já inseriu na máquina para determinada compra);
  - Total (representando o total de dinheiro já inserido na máquina, desde que ela foi ligada, até o atual momento);

# Exemplo 2 – Máquina de Passagens

- Escreva métodos que representem a inserção de dinheiro e a emissão de passagens
  - Lembre-se que a máquina só emitirá a passagem caso o dinheiro total correspondente ao preço da passagem seja inserido
  - Caso o dinheiro inserido não seja suficiente, a máquina deve avisar ao usuário quanto de dinheiro falta para ser inserido
  - Assim que o dinheiro total é inserido, a máquina automaticamente emite a passagem
- Escreva métodos que permitam à máquina mostrar ao usuário o preço da passagem

# Exemplo 2 – Máquina de Passagens

- Problema: como representar no sistema que o preço da passagem é R$2,00?
- Duas maneiras:
  - Fixo no código: solução ruim!
  - Atributo: flexibilização
- Construtor pode auxiliar

# Exemplo 2 – Máquina de Passagens

```
public class MaquinaPassagens {

    private int preco;
    private int quantidadeInserida;
    private int total;

    public MaquinaPassagens() {     {
        preco = 2;
        quantidadeInserida = 0;
        total = 0;
    }

    //Metodos...

}
```

# Exemplo 2 – Máquina de Passagens

- Observe que o construtor pode ser melhorado
- Do jeito que está, o preço das passagens será sempre R$2,00
- E se quisermos criar várias máquinas, cada uma com um preço diferente para a passagem?

# Exemplo 2 – Máquina de Passagens

```
public class MaquinaPassagens {

    private int preco;
    private int quantidadeInserida;
    private int total;

    public MaquinaPassagens(int p) {
        preco = p;
        quantidadeInserida = 0;
        total = 0;
    }

    //Metodos...

}
```

# Atributos final

- Um campo final deve ser inicializado quando o objeto é construído e, posteriormente, não pode ser modificado. Ex:

```
public class Employee{
    private final String name;
     ...
}
```

- É útil para atributos cujo tipo é primitivo ou classes imutáveis (nenhum dos seus métodos transforma seus objetos)

# Atributos e Métodos Estáticos

- Atributo estático
  - Há apenas um desses atributos por classe individual. Ex:

```
public class Employee{
   private int id;
   private static int nextId = 1
}
```

  - Se houver 100 objetos Employee, haverá 100 atributos *id*, uma para cada objeto, mas apenas um campo estático *nextId*

# Atributos e Métodos Estáticos

- Constante estática
  - Variáveis estáticas são raras; constantes estáticas são comuns. Ex:

```
public class Math{
   public static final double PI=3.1415...;
   ...
}
```

  - Você pode acessar a constante Math.PI

# Atributos e Métodos Estáticos

- Métodos estáticos
  - São aqueles que não operam em objetos
  - Não podem acessar atributos da classe, somente os estáticos
  - Ex: método pow da classe Math
    - Math.pow(x,a)
    - Calcula a potência x^a. Não utiliza nenhum objeto Math, ou seja, não tem nenhum parâmetro implícito

# Atributos e Métodos Estáticos

- Métodos estáticos – Quando usar
  - Quando um método não precisa acessar o estado de objeto porque todos os parâmetros necessários são fornecidos como parâmetro explícito
  - Quando um método só precisa acessar campos estáticos da classe

    - **Por que o método main é estático?**

# Referências


- DEITEL, H. M., DEITEL, P. J., **Java: Como Programar,** Bookman, São Paulo, 2002
- BARNES, D. J., KOLLING, M., **Programação Orientada a Objetos com Java**, 2004, ISBN: 8576050129