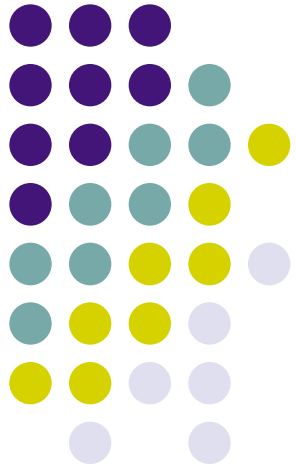
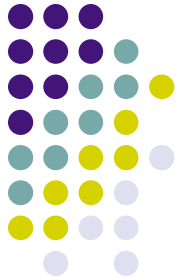
POO com Java

Construtores e Sobrecarga

Rita Suzana



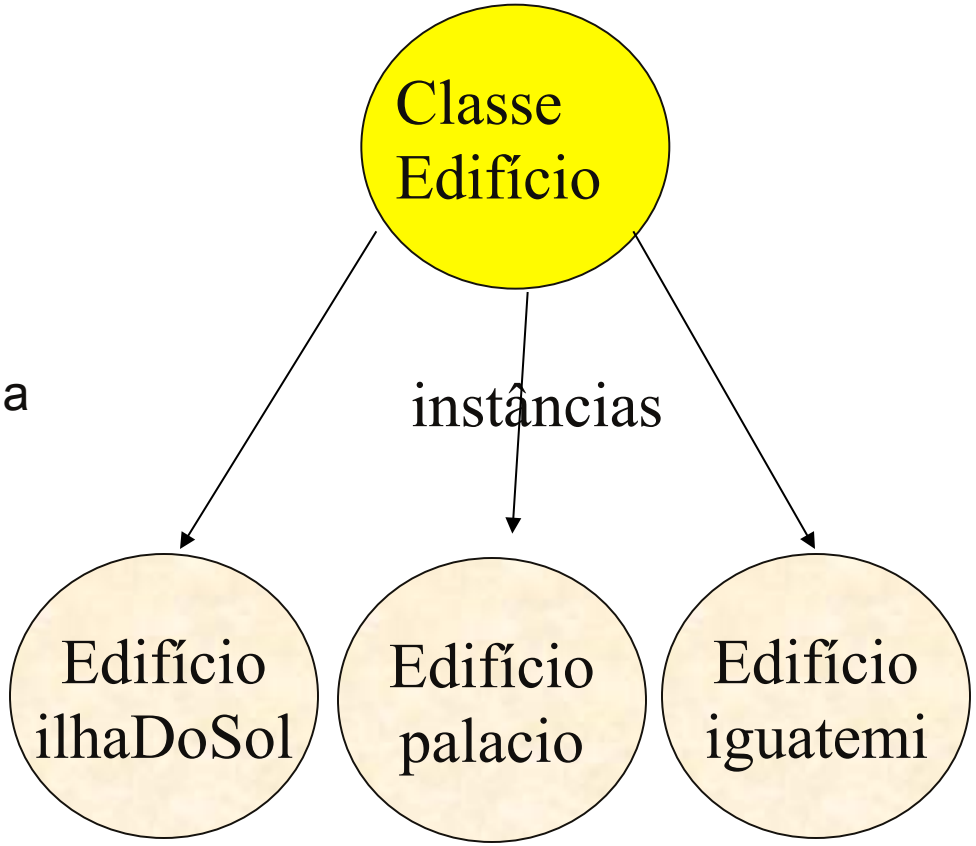
Agenda



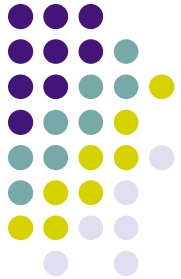
- Referência
- Construtores
- Sobrecarga
- This

Classes e Objetos

- Para que *objetos* ou *instâncias* possam ser manipulados, é necessária a criação de *referências* a estes objetos, que são variáveis do "tipo" da classe.
- Exemplo
  - Classe -> Planta do edifício, que descreve o edifício, mas não corresponde fisicamente a ele.
  - Instância -> Edifício construído.
  - Referência -> Nome do Objeto (ilhaDoSol)

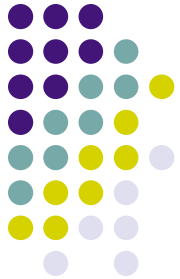


Construtores



- Até o momento, criamos instâncias da seguinte maneira:
  - Criamos um método na classe com o nome da classe
  - No blueJ
    - Chamamos o comando **new**

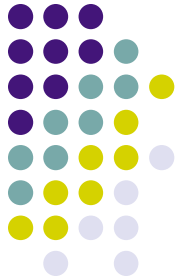
Construtores



- Construtor é um tipo especial de membro de classe chamados automaticamente quando instâncias são criadas através da palavra chave **`new`**
  - Programador não pode chamar construtores diretamente;

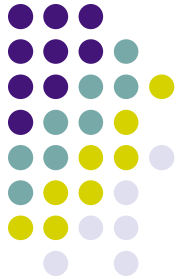
- Construtores são úteis para:
  - inicializar os atributos de uma classe
  - realizar rotinas complexas de inicialização
  - realizar inicializações segundo parâmetros passados no momento da criação do objeto

Construtores



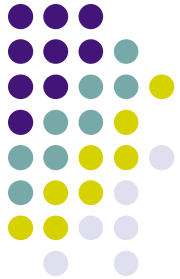
- Na declaração de um construtor deve-se considerar que:
  - Construtores devem ter **exatamente** o mesmo nome da classe
  - Construtores não possuem tipo de retorno
  - Construtores são, normalmente, públicos

Construtores



- Toda classe possui um construtor
- Caso o programador não forneça uma versão explícita do construtor...
  - Java utiliza uma versão de construtor implícita que não recebe nenhum parâmetro
- Se o programador fornecer qualquer versão de um construtor...
  - a linguagem não mais incluirá o construtor padrão implícito

Construtores



- Exemplo

```
public class Ponto{
  private float x;
  private float y;
  public Ponto(float pX,float pY){
    x = pX;
    y = pY;
   }
  }
```

Exemplo Construtor

```
public class Empregado
{  String Nome, Endereço, CPF;
    float Salário;
    int Idade;

   public Empregado ( String N, String End, String cpf, int Ida)
   {   Nome = N;
       Endereço = End;
       CPF = cpf;
       Idade = Ida;
   }

   public String  InformarCPF( )
   {   return CPF;
   }

   public String ApresentarNome( )
   {  return Nome;
   }
}
```

Construtor da classe Empregado. Observe que não foi definido um tipo de retorno e possui o mesmo nome da classe.
Modificador de acesso é public

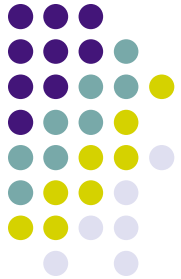
Construtores

Utilização do Construtor da classe Empregado para instanciar um objeto.

```
class CriaEmpregado
{  public static void main(String args[])
   {
            Empregado obj = new Empregado(“Maria Silva”, “Rua X  nº
                                                    246", "403.234.567-33",
58);
      System.out.println("O nome é  = ” +obj.ApresentarNome());

       System.out.println("O CPF é =  "+ obj.InformarCPF());

       System.out.println("A idade é = “+ obj.DevolveIdade());
   }
}
```

Nome da classe

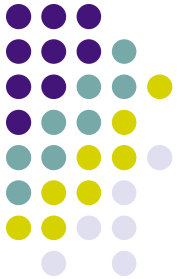
Nome do Variavel



Primitivas e Referências

- Variáveis de tipos primitivos armazenam um valor
  - Na declaração aloca-se espaço na memória suficiente para o armazenamento da variável.

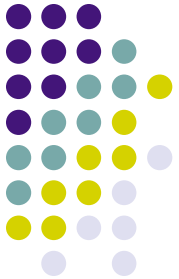
- Variáveis de tipos referências armazenam identificadores para objetos
  - Na declaração aloca-se espaço para a referência ao objeto.
  - A alocação do objeto é realizada através do operador new.



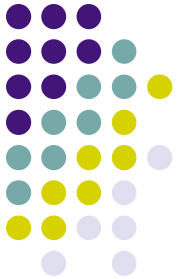
Primitivas e Referências

- **new** instancia um novo objeto.
- **new** aloca o espaço necessário para armazenar o estado do objeto e uma tabela para indicar o corpo do código necessário para efetuar suas operações.
- **new** executa as tarefas de inicialização do objeto conforme o seu Construtor.
- **new** 'retorna' o identificador do objeto criado e o operador de atribuição é utilizado para copiar este endereço para uma variável, que servirá para futuras referências ao objeto.
- O compilador e a máquina virtual Java verificam o tipo da variável e o tipo da referência retornada

Construtores



- E no exemplo do triangulo?
- E no exemplo da locadora?
  - Como ficaria a criação dos filmes e cliente?
    - Pacote locadora completo



Sobrecarga

- O que acontece se quisermos construir um objeto de diferentes maneiras?
- Se precisarmos de diferentes maneiras de se instanciar um objeto da mesma classe?
- Precisamos escrever e permitir que a classe disponibilize diversos construtores.
- Construtores com os tipos dos parâmetros diferentes.
  - Ex:

```
public Casa(int num, String cor, int qtdQuartos){...

public Casa(int num, int qtdQuartos){...}

public Casa(int num, byte qtdQuartos){...}
```

Sobrecarga

- Exemplo

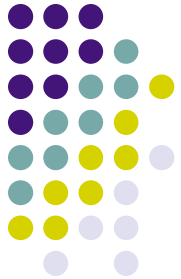
```
public Automovel(String tipoComb, String fabricante,
String modelo, int ano){...}
```

Talvez não seja necessário, em alguns casos, poder instanciar um carro com diversos tipos de combustível.

Ou talvez essa informação (tipo de combustível) só deva ser atribuída mais tarde no decorrer da execução do programa.

```
public Automovel(String fabricante, String modelo,
int ano){...}
```

Sobrecarga

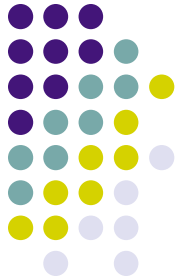


- Ex

Talvez queiramos uma forma de instanciar objetos automóveis sempre com o ano atual. Logo, não precisamos passar como parâmetro o ano. Simplesmente, atribuímos internamente, sempre o ano atual.

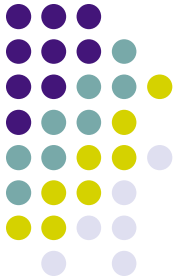
```
public Automovel(String tipoComb, String fabricante,
String modelo)
{
    ...
    ano = obter ano atual do sistema;
    ...
}
```

Sobrecarga



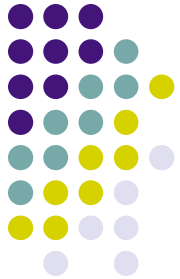
- Um método pode ter o mesmo nome que outro método na mesma classe.
- Isto é usual quando existem duas ou mais formas diferentes de se realizar a mesma tarefa
- Neste caso diz-se que o método está sobrecarregado

Sobrecarga



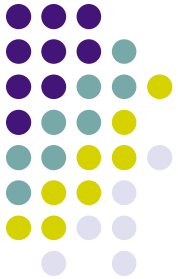
```
public class Ponto{
  private float x;
  private float y;
  public Ponto(float pX,float pY){
    x = pX;
    y = pY;
   }
  public void mover(float novoX,float novoY){
    x = novoX;
    y = novoY;
  }
  public void mover(){
    x = 0;
    y = 0;
  }
}
```

Sobrecarga



- Uma sobrecarga é válida se a assinatura do método difere no número ou no tipo dos parâmetros
- Java permite a criação de métodos com nome iguais, contanto que as assinaturas sejam diferentes;
- A assinatura é composta do nome mais os tipos de argumento. O tipo de retorno não faz parte da assinatura;
  - Diferenças do tipos dos parâmetros é que definem assinaturas diferentes.
  - Diferenças no nome dos parâmetros não são relevantes
  - Diferenças no tipo de retorno são permitidas, mas não são relevantes

Sobrecarga



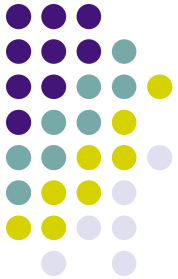
- Definiremos então, métodos com o mesmo nome, porém assinaturas diferentes.
  - Assinatura: nome do método + tipo dos argumentos passados como parâmetro
  - Atenção: o tipo de retorno do método não faz parte da assinatura.

É um exemplo de sobrecarga?

```
public void pintarCasa(String c)
{...}

public void pintarCasa2(String c)
{...}

public void pintarCasa3(String c)
{...}
```

Não! São três métodos diferentes independente do que fazem internamente.

Sobrecarga



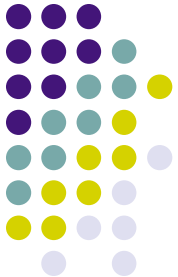
- Assinatura: nome do método + tipo dos argumentos pasados como parâmetro
- Atenção: o tipo de retorno do método não faz parte da assinatura.

```
public void pintarCasa(String c)
{
    wcor = c;
}

public String pintarCasa(String cor)
{
    wcor = cor;
    return cor;
}
```

É um exemplo de sobrecarga?

Não! As assinaturas são iguais: mesmo nome do método e mesmo tipo do parâmetro.
Erro de compilação!

Sobrecarga



- Assinatura: nome do método + tipo dos argumentos pasados como parâmetro
- Atenção: o tipo de retorno do método não faz parte da assinatura.

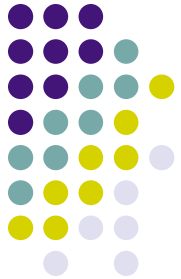
```
public void pintarCasa(String c)
{
    cor = c;
}

public void pintarCasa()
{
    cor = "branco";
}
```

É um exemplo de sobrecarga?

Sim! Assinaturas diferentes: nomes iguais, porém parâmetros diferentes.

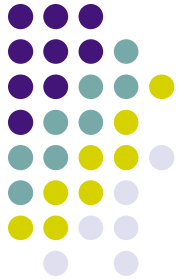
Podemos ter sobrecarga de diversos métodos diversas vezes.

Sobrecarga



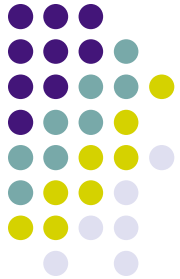
- Java admite também, que se sobrecarregue os construtores de uma classe.
- As restrições são similares àquelas aplicadas aos métodos sobrecarregados
- Pode-se se referir de dentro de um construtor sobrecarregado para outro, através do uso da palavra reservada **`this`**

A referência this



- Na chamada dos métodos de uma classe, a máquina virtual passa implicitamente como parâmetro uma referência ao objeto ao qual a mensagem foi enviada.
- Quando se deseja recuperar esta referência de dentro do corpo da classe, usa-se a palavra reservada `this`

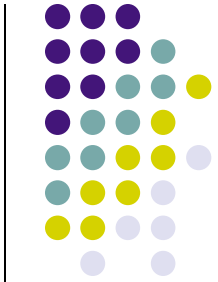
A referência this



- A referência this é usada para:
  - referência ao próprio objeto
  - acesso a variáveis dos objetos
  - passagem do objeto como parâmetro

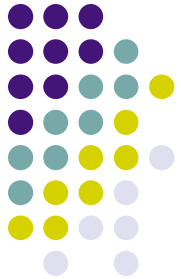
```
public class Circulo{//classe Círculo
  private float raio = 0;
  public void alterarRaio(float raio){
    this.raio = raio;
  }
}
```

Automóvel com this



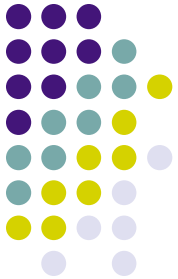
```
public class automovel
{
   private String tipoCombustivel;
   private String fabricante;
   private int ano;
   private int modelo;

   public automovel(String tipoCombustivel, String fabricante, int ano, int modelo)
   {
      this.tipoCombustivel = tipoCombustivel;
      this.fabricante = fabricante;
      this.ano = ano;
      this.modelo = modelo;
    }

   public float calculaConsumo(int percurso)
   {
      float taxaDeConsumo = -1f;
      if (this.tipoCombustivel=="gasolina")
      {
         taxaDeConsumo = 15;
       }
       else
           if (this.tipoCombustivel=="alcool")
           {
              taxaDeConsumo = 12;
           }
           else
              if (this.tipoCombustivel=="flex")
              {
                 percurso = 3*percurso;
                 taxaDeConsumo = 40;
```

Sobrecarga e this



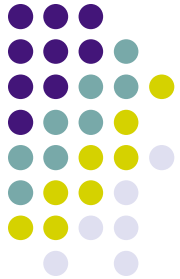
- Pode-se ainda utilizar a palavra chave **`this`** para se referir a um construtor sobrecarregado
- Entretanto, neste caso, a palavra chave não diz respeito à referência do objeto, visto que este ainda está sendo criado

Sobrecarga e this



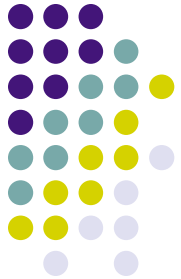
```
public class Circulo{//classe Círculo
  private Ponto origem;
  private float raio;
  public Circulo(){
    this(0,0,0); // this(new Ponto(0,0), 0);
  };
  public Circulo(float x, float y, float raio){
    this(new Ponto(x,y), raio);
  };
  public Circulo(Ponto p, float raio){
    this.origem = p;
    this.raio = raio;
  }
}
```

Exercício



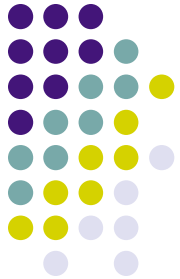
- Criar uma classe que represente um triangulo. Esta classe possuirá os atributos a, b e c (lados do triângulo). Caso o objeto seja instanciado apenas com um parâmetro, o triângulo deverá ser eqüilátero (três lados iguais), caso o objeto seja instanciado com dois parâmetros, o triângulo deverá ser isósceles e o primeiro parâmetro indicará o tamanho de a e b, caso haja três parâmetros na iniciação do objeto, o triângulo é escaleno.
  - Faça também um método para calcular o perímetro de um triangulo. Caso não seja passado nenhum parâmetro, o método devolve o perímetro do triângulo já instanciado. Caso os três lados sejam passados será o perímetro do triangulo que possui estes três lados.

Get e Set



- Get: utilizados para obter o valor do atributo.
- Set: utilizado para alterar o valor de um atributo.
- Usado para manter um único ponto de entrada

Get e Set



```
public class Circulo{//classe Círculo
  private Ponto origem;
  private float raio;
  public Circulo(){
    this(0,0,0); // this(new Ponto(0,0), 0);
  };
  public Circulo(float x, float y, float raio){
    this(new Ponto(0,0), 0);
  };
  public Circulo(Ponto p, float raio){
    setPonto(p);
    setRaio(raio);
  }
  public Ponto getOrigen(){
      return origem;
    }
  public void setPonto(Ponto p){
      this.origem = p;
    }
  public float getRaio(){
       return raio;
    }
  public void setRaio(float r){
      this.raio = r;
    }
}
```