Universidade Federal do Rio Grande do Sul
Departamento de Informática Aplicada

Programação Orientada a Objeto

10. Construtores, Atributos e Métodos de Classe e Constantes (Prática)

Prof. Dr. Thiago L. T. da Silveira
tltsilveira@inf.ufrgs.br

2° Semestre de 2020

Objetivos

- Exercitar os conceitos de POO vistos na aula anterior!
  - Escrevendo, compilando e interpretando (executando) aplicações em Java;
  - Construtores, Atributos e Métodos de Classe e Constantes.

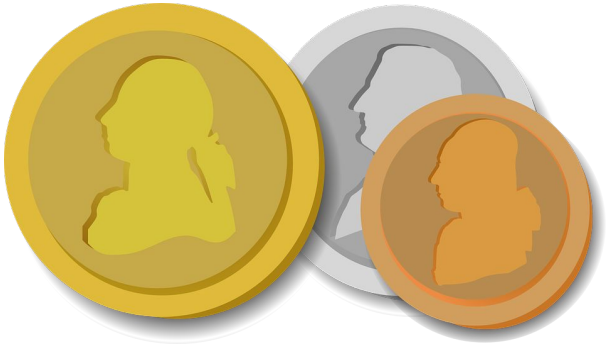
Exercício #1 Parte #0

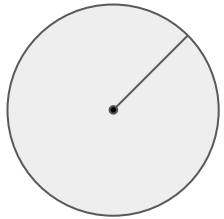
- Uma empresa terceiriza serviços para um banco que tem de, diariamente, conferir a quantidade e valores das moedas que chegam às suas instalações. Para essa tarefa, a empresa usa um algoritmo de Visão Computacional que captura imagens, identifica círculos e conta a quantidade dessas primitivas nas imagens. No entanto, algoritmo encontra dificuldades quando as moedas estão encostadas umas nas outras ou sobrepostas. Com essa abstração em mente, [...]

Imagens livres de https://www.pixabay.com/

Exercício #1 Parte #1

- Criar uma classe em Java (**`Circulo`**) que abstrai a detecção de uma moeda na imagem e contém propriedades e operações conforme especificação abaixo:





Circulo
- double x - double y - double raio - int identificador -* int numCirculos -*^ double PI
+ Circulo(double, double, double) + void move(String, double) + double area() + double circunferencia() + void imprime() + int getIdentificador() +* boolean sobreposicao(Circulo, Circulo)

Imagens livres de https://www.pixabay.com/

Exercício #1 Parte #1

- Definir os atributos e métodos conforme notação: (-) privados, (+) públicos, (*) atributos/métodos de classe e (^) constantes;
- Aproxime π com quatro casas decimais;
- Inicialize `numCirculos` com o valor zero;
- O construtor deve receber três parâmetros do tipo `double` referentes aos atributos `x`, `y` e `raio`, nessa ordem; O construtor incrementa o atributo `numCirculos` e atribui esse valor a `identificador` [...];

Circulo
- double x - double y - double raio - int identificador -* int numCirculos -*^ double PI
+ Circulo(double, double, double) + void move(String, double) + double area() + double circunferencia() + void imprime() + int getIdentificador() +* boolean sobreposicao(Circulo, Circulo)

Exercício #1 Parte #1

- O método **`move`** recebe **`String direcao`** e **`double deslocamento`**; Esse método incrementa **`x`** se **`direcao`** for “direita” e **`y`** se **`direcao`** for “cima” ou decrementa **`x`** se **`direcao`** for “esquerda” e **`y`** se **`direcao`** for “baixo”, respectivamente; Os incrementos e decrementos se dão em **`deslocamento`** unidades;
- Os métodos **`area`** e **`circunferencia`** calculam a área (*A*) e a circunferência (*C*) de um círculo centralizado em (*x*, *y*) com raio *r* da seguinte forma:

$A = \pi.r^2$ e $C = 2.\pi.r$

Circulo
- double x - double y - double raio - int identificador -* int numCirculos -*^ double PI
+ Circulo(double, double, double) + void move(String, double) + double area() + double circunferencia() + void imprime() + int getIdentificador() +* boolean sobreposicao(Circulo, Circulo)

Exercício #1 Parte #1

- O método **`imprime`** mostra uma mensagem na tela. Se o círculo tem **`identificador`** 1, **`raio`** 5 e é centralizado em **`x`** = 1 e **`y`** = 2, então a mensagem impressa será "C1 = {(1.0, 2.0), 5.0}";
- O método **`getIdentificador`** retorna o valor de **`identificador`**;
- O método **`sobreposicao`** indica se dois objetos da classe **`Circulo`** se tocam ou se sobrepoẽm; Isso é verdade quando dois círculos centrados em (x_1, y_1) e (x_2, y_2) com raios r_1 e r_2 satisfazem a seguinte inequação

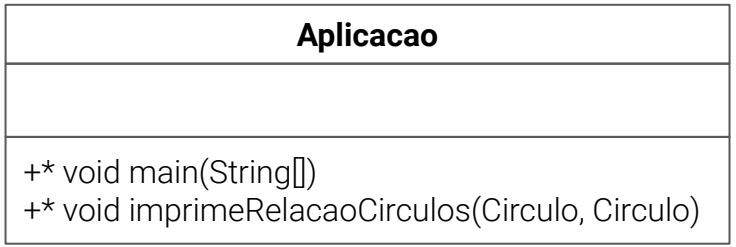
$$(x_1-x_2)^2+(y_1-y_2)^2 \leq (r_1 + r_2)^2$$

Circulo
- double x - double y - double raio - int identificador -* int numCirculos -*^ double PI
+ Circulo(double, double, double) + void move(String, double) + double area() + double circunferencia() + void imprime() + int getIdentificador() +* boolean sobreposicao(Circulo, Circulo)

Exercício #1 Parte #2

- Criar uma outra classe em Java (**`Aplicacao`**) que representa o programa principal, conforme especificação ao lado.
- Crie e instancie dentro do método ***`main`*** quatro objetos da classe **`Circulo`** (`cA`, `cB`, `cC` e `cD`) com os dados abaixo:

Objeto	Atributo `x`	Atributo `y`	Atributo `raio`
`cA`	1	1	2
`cB`	3	0	3
`cC`	2	2	2
`cD`	-5	4	5



Exercício #1 Parte #3

- Crie um método **`imprimeRelacaoCirculos`** de **`Aplicacao`** que recebe dois objetos da classe **`Circulo`** e tem o seguinte corpo:

```
public static void imprimeRelacaoCirculos(Circulo c1, Circulo c2){
       if (Circulo.sobreposicao(c1, c2))
          System.out.println("C"+c1.getIdentificador()+" e C"+c2.getIdentificador()+"
se sobrepõem");
       else
          System.out.println("C"+c1.getIdentificador()+" e C"+c2.getIdentificador()+"
não se sobrepõem");
   }
```

Exercício #1 Parte #4

- Dentro do método **`main`** de **`Aplicacao`** chame o método **`imprime`** de cada objeto;
- Na sequência, chame **`imprimeRelacaoCirculos`** seis vezes, passando como argumento **`cA`** e **`cB`**, **`cA`** e **`cC`**, **`cA`** e **`cD`**, **`cB`** e **`cC`**, **`cB`** e **`cD`** e **`cC`** e **`cD`**, respectivamente;
- Quais pares de círculos se sobrepõem?

Exercício #1 Parte #5

- Dentro do método **`main`** de **`Aplicacao`** chame o método **`move`** da seguinte forma:
  - Mova **`cA`** para “baixo” e para “esquerda” em 1 unidade, cada;
  - Mova **`cB`** para “direita” e para “cima” em 6 e 5 unidades, respectivamente;
  - Mova **`cC`** para “cima” em 2 unidades;
  - Mova **`cD`** para “cima”, “esquerda”, “baixo” e “esquerda” em 2, 3, 1 e 1 unidades, respectivamente;
- Na sequência, chame o método **`imprime`** de cada objeto;
- Na sequência, chame novamente **`imprimeRelacaoCirculos`** seis vezes, passando como argumento **`cA`** e **`cB`**, **`cA`** e **`cC`**, **`cA`** e **`cD`**, **`cB`** e **`cC`**, **`cB`** e **`cD`** e **`cC`** e **`cD`**, respectivamente;
- Quais pares de círculos se sobrepõem?

Exercício #1 - Exemplo de Execução (com **outros círculos!**)

```
C1 = {(2.0, 0.0), 3.0}
C2 = {(0.0, -1.0), 3.0}
C3 = {(3.0, 3.0), 5.0}
C4 = {(-2.0, 1.0), 2.0}
C1 e C2 se sobrepõem
C1 e C3 se sobrepõem
C1 e C4 se sobrepõem
C2 e C3 se sobrepõem
C2 e C4 se sobrepõem
C3 e C4 se sobrepõem
C1 = {(1.0, -1.0), 3.0}
C2 = {(6.0, 4.0), 3.0}
C3 = {(3.0, 5.0), 5.0}
C4 = {(-6.0, 2.0), 2.0}
C1 e C2 não se sobrepõem
C1 e C3 se sobrepõem
C1 e C4 não se sobrepõem
C2 e C3 se sobrepõem
C2 e C4 não se sobrepõem
C3 e C4 não se sobrepõem
```

Atividades

- Entrega de um arquivo **.zip** (contendo a estrutura de diretórios da aplicação e os dois arquivos .java)
  - Entregue a implementação do Exercício #1 Partes #1 a #5
  - Indique com um comentário (“//” ou “/**/”) o que acontece nos Exercício #1 Partes #4 e #5
  - Entrega até às 23:55h de 04/03/2021

Universidade Federal do Rio Grande do Sul
Departamento de Informática Aplicada

Programação Orientada a Objeto

10. Construtores, Atributos e Métodos de Classe e Constantes (Prática)

Prof. Dr. Thiago L. T. da Silveira
tltsilveira@inf.ufrgs.br

2° Semestre de 2020