

## INF 213 - Roteiro da Aula Prática 2

O objetivo desta aula é praticar a criação de classes utilizando a técnica de herança.

Arquivos fonte e diagramas utilizados nesta aula:

<https://drive.google.com/open?id=1D8lg73W61Nu25rgEFT6Qej0qTaU5X8Cx>

### Etapa 1

Obtenha os diagramas UML das classes *Retangulo* e *Circulo* e as implementações dadas nos arquivos *Retangulo.h*, *Retangulo.cpp*, *Circulo.h* e *Circulo.cpp*, junto com o programa teste **TesteFiguras.cpp**. Analise as implementações baseadas nos diagramas UML e execute o programa teste.

### Etapa 2

Refaça a implementação das classes *Retangulo* e *Circulo* utilizando o conceito de herança baseando-se no diagrama de classes UML dado no arquivo **RetanguloCirculoComHerancaUML.pdf**. Teste a sua implementação com o programa **TesteFiguras.cpp** (o mesmo anterior) e confira o resultado obtido com o resultado esperado incluído no código do programa.

Dicas:

- não se esqueça de declarar os métodos que não modificam os objetos como constantes.
- se a classe B herda da classe A (de forma pública), na declaração de B (no cabeçalho da classe) utilize a sintaxe "class B: public A {".
- lembre-se de chamar o construtor de *FigBase* de forma apropriada nos construtores das classes derivadas. Como *FigBase* não possui construtor padrão, se o construtor não for chamada de forma explícita haverá um erro de compilação (visto que o compilador tentará chamar o construtor padrão).

### Etapa 3

Implemente uma classe *Segmento*, como uma subclasse (classe derivada) de *FigBase* conforme definido no diagrama de classes

**RetanguloCirculoSegmentoComHerancaUML.pdf**. Execute o programa **TesteFiguras2.cpp** e confira o resultado obtido com o resultado esperado incluído no código do programa.

Obs: os conceitos de área/perímetro não fazem muito sentido em Segmentos.... Como resolver esse problema?

### Etapa 4

Inclua a sobrecarga dos operadores de entrada (>>) e saída (<<) nas classes *Retangulo*, *Circulo* e *Segmento*. Para testar, rode o programa **TesteFiguras3.cpp** e digite os dados dos objetos lidos.

Para evitar a duplicação de código, crie funções auxiliares ("le", "imprime") em todas as classes da hierarquia e utilize tais funções nos operadores de entrada e saída. Lembre-se que a implementação dessas funções nas classes derivadas deve chamar as versões da classe básica (para reaproveitar melhor o código).

Por simplicidade, supomos que as funcoes de leitura e impressao sempre leem os dados do “cin” e escrevem a saida no “cout” (suponha tambem que os operadores << e >> sempre trabalham com os streams *cin* e *cout*). Assim, embora os operadores << e >> receberao um stream como argumento, tal stream não será utilizado (o cin/cout será utilizado no lugar dele). Uma versao mais generica (e correta) da funcao imprime, por exemplo, deveria receber como argumento um stream (*ostream*) e utiliza-lo no lugar do *cout*. (como exercicio opcional, faca com que as funcoes “le” e “imprime” usem esses streams mais genericos).

Obs: adicione valores-padrao para os argumentos dos construtores de Retangulo, Circulo e Segmento pois antes de ler valores para objetos de tais classes os objetos serao criados utilizando o construtor sem argumentos.

### **Submissao da aula pratica:**

A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submittty ([submittty.dpi.ufv.br](http://submittty.dpi.ufv.br)).

Envie pelo sistema Submittty apenas a versão final de suas classes e cabecalhos (arquivos .h) (obtidas após o término da etapa 4).

Obs: nesta aula pratica o formato da entrada/saida não sera avaliado (i.e., não ha nenhuma regra rigida sobre a ordem e o formato dos dados lidos e impressos em tela).