Programação Orientada a Objetos e Estrutura de Dados

Ementa:

- **Unidade I**
  - Introdução à orientação a objetos.
  - Linguagens típicas orientadas a objetos.
  - Programação orientada a objeto.
  - Conceitos básicos e terminologias de programação orientada a objetos.
  - Classe, objetos, atributos, métodos, construtores e sobrecarga.
  - Instanciação e referência de objetos.
  - Envio de mensagens.
  - Ciclo de vida de um objeto.
  - Abstração e encapsulamento.
- **Unidade II**
  - Herança.
  - Criação e uso de hierarquia de classes.
  - Classes abstratas e interfaces.
  - Relacionamento entre classes.
  - Polimorfismo.
  - Ligação dinâmica (*dynamic bindgind*).
  - Tratamento de exceções.
- **Unidade III**
  - Listas: estática sequencial / encadeada.
  - Listas: dinâmica simples / duplamente encadeada.
  - Listas: aplicações.
- **Unidade IV**
  - Filas.
  - Pilhas.
  - Árvores.
  - Técnicas de ordenação.
  - Técnicas de pesquisa.

Vai ser um pouco puxado, mas nós CONSEGUIREMOS!

Linguagens: Java e C/C++

Revisão (para quem já veio do primeiro período) e Primeiro Contato para os novos alunos

- **O que é um algoritmo?**
  - Grosso modo, são instruções passo-a-passo para realizar alguma tarefa.
  - Ex.: Algoritmo para enviar mensagem para o colega:
    - (1) Desbloqueie seu celular;
    - (2) Procure o aplicativo de mensagens;
    - (3) Abra o aplicativo;
    - (4) No aplicativo, procure pelo contato desejado;
    - (5) Abra o chat com o contato selecionado;
    - (6) Escreva uma mensagem;
    - (7) Pressione o botão de enviar.
- **Como um computador entende um algoritmo?**
  - Primeiramente um programador escreve o algoritmo utilizando alguma linguagem de programação.
  - Ao finalizar, o programador utiliza outro programa: compilador/interpretador.
  - O compilador/interpretador vai transformar o seu código em instruções que o computador entende.
- **Como eu posso fazer um algoritmo?**
  - Em primeiro lugar, lembre-se que um computador é um “escravo burro”, ou seja, ele vai fazer exatamente tudo o que você mandar, sem sequer pensar se aquilo é bom ou ruim, se vai dar certo ou errado, etc. Se ele não souber o que você mandou fazer, ele simplesmente não faz.
  - Você pode começar desenhando algum diagrama de fluxo, ou identificando os pontos-chave de seu algoritmo.
  - Com isso você terá em mãos a lógica de como seu algoritmo deve funcionar.
  - Depois escolha uma linguagem de programação, e transcreva nela a sua lógica.
    - A lógica é comumente referida como a **semântica** do algoritmo/programa, enquanto que a forma como você a implementa é referida como **sintaxe**.
- **Como programar?**
  - Veremos em detalhes com Java e C/C++ durante o curso.
  - Porém, todas as linguagens possuem alguns termos e sintaxes comuns.
  - Praticamente todas as linguagens têm sua sintaxe em **Inglês**.

    - **Sintaxe comum (igual ou muito parecido entre todas as linguagens)**
        - Tipos de dados.
        - Operadores.
        - Controle de fluxo.
        - Estruturas de Dados.
- **Tipos de dados**
    - Quando criamos algoritmos nós comumente manipulamos variados tipos de dados.
    - Todas as linguagens têm seus tipos de dados possíveis, chamados **Tipos Primitivos**. Serão apresentados aqui os mais comuns.
        - **Tipos numéricos**
            - **Inteiro**
                - É o tipo mais comum.
                - Os hardwares suportam vários tamanhos de inteiros (em bits).
                - Cada linguagem tem suporte nativo para alguns ou todos esses tamanhos.
                - Tipos comuns: **int**, **short**, **long**, e também suas versões sem sinal, com a palavra **unsigned** antes do tipo.
            - **Ponto flutuante**
                - É o tipo de dados que modela os números reais, ou a aproximação de determinados valores, por exemplo, o valor de π.
                - São valores como 1,25 ou 1,64634 e assim por diante. Quando você estiver programando, lembre-se de que a língua usada é o Inglês, então em vez de 1,25 será 1.25.
            - **Complexo**
                - A maioria das linguagens não dão suporte nativo a números complexos. O programador, entretanto, pode se utilizar de seus conhecimentos de **estruturas de dados** para criar algum tipo de dado derivado.
                - Os valores complexos são comumente representados por pares ordenados de valores de ponto flutuante. Por exemplo (2, 3) pode ser transformar no número 2 + 3i.
                - Cada linguagem que suporta esse tipo de dado tem sua forma de representá-lo. Então você precisa ver a documentação de cada linguagem para saber como utilizar números complexos lá.
        - **Tipos booleanos**
            - É, possivelmente, o tipo mais simples.

        - Existem apenas dois valores *booleanos*, o valor **verdadeiro** e o valor **falso**. Linguagens que suportam esse tipo possuem as palavras reservadas **True** e **False**. Caso contrário, você pode simular com os valores 0 (para falso) e 1 (para verdadeiro).
    - **Caracteres**
        - Dados na forma de caracteres são armazenados como codificações numéricas. Tradicionalmente a codificação mais utilizada era a ASCII de 8 bits.
        - Atualmente o UTF-8 vem se tornando o padrão.
        - Para utilizar um dado do tipo caractere, as linguagens pedem que você torne isso explícito com a palavra reservada **char**.
    - **Cadeias de Caracteres**
        - Uma palavra, ou uma frase, é uma **cadeia de caracteres**. Em linguagens mais antigas é um pouco complicado de manipular. Nas linguagens mais novas, Java incluso, podemos manipular através de um tipo chamado **String**.
- **Operadores**
    - Operadores são, grosso modo, os símbolos que utilizamos para executar alguma inferência sobre tipos de dados. Existem vários tipos de operadores.
    - **Operadores aritméticos**
        - Soma
            - Com o operador **+**.
        - Subtração
            - Com o operador **-**.
        - Multiplicação
            - Com o operador *****.
        - Divisão
            - Com o operador **/**.
        - Módulo
            - Retorna o resto da divisão;
            - Com o operador **%**.
        - Expoente
            - Eleva um número a outro.
            - Normalmente o operador é ****** ou **^**.
    - **Operadores de atribuição**
        - É o operador que atribui um valor a uma **variável**.
        - É o operador **=**.

        - Algumas linguagens podem juntar o operador de atribuição com operadores aritméticos. Ex.: x += 1, faz com que ao valor de x seja somado o valor 1.
    - **Operadores relacionais**
        - Verificam a comparação entre dois operandos e retornam um valor *booleano*, ou seja, verdadeiro ou falso.
        - Operador maior que: >.
        - Operador menor que: <.
        - Operador igual a: ==.
        - Operador diferente de: != ou ~=.
        - Operador maior ou igual que: >=.
        - Operador menor ou igual que: <=.
        - Exemplo: supondo x = 4 e y = 5.
            - x > y : falso.
            - x < y : verdadeiro.
            - x == y : falso
            - x != y : verdadeiro.
    - **Operadores lógicos**
        - Realizam a operação lógica entre dois operandos (normalmente condições) e retornam valores *booleanos*.
        - Operador **E** (and, &&). x **E** y é verdadeiro somente se x e y forem verdadeiros.
        - Operador **OU** (or, ||). x **OU** y é verdadeiro se qualquer dos dois operandos for verdadeiro.
        - Operador **NÃO** (not, ~, !). Reverte ou complementa o operando.
    - **Operadores binários**
        - São operadores lógicos, porém atuam bit por bit. O bit 1 significa **verdadeiro** e o bit 0 significa **falso**. Por exemplo, vamos assumir que x = 5 = 0101 e y = 7 = 0111.
        - Operador **E** (AND, &).
            - x & y: 0101 & 0111: 0101
        - Operador **OU** (OR, |)
            - x | y: 0101 | 0111: 0111.
        - Operador **XOU** (XOR, ^)
            - x ^ y: 0101 ^ 0111: 0010.
        - Operador **NÃO** (NOT, ~)
            - ~ x: ~ 00000101: 11111010.
        - Operador **Right Shift** (»)

        - x » 1: 0101 » 1: 0010.
    - Operador **Left Shift** («)
        - x « 1: 0101 « 1: 1010.
- **Controle de fluxo**
    - O fluxo de um programa pode ser controlado/manipulado a partir de **blocos de controle** e **blocos de repetição**.
    - **Blocos de Controle**
        - **If-Else**
            - O bloco de controle mais conhecido. Sua estrutura básica:
                - **se** condição
                    - comandos
                - **senão se**
                    - comandos
                - **senão**
                    - comandos
            - Como as palavras reservadas em programação são em Inglês, o **se** é **if**, **senão se** é **else if**, e **senão** é **else.**
        - **Switch** ou **Match**
            - Dependendo da situação, você quer comparar o valor de uma variável a múltiplos valores específicos. Para cada valor que a variável puder assumir, há um conjunto diferente de comandos/instruções.
            - Ex.:
                - **switch** (variável)
                    - **case 1**
                        - comandos
                    - **case 2**
                        - comandos
                    - **case 3**
                        - comandos
                    - **default**
                        - comandos
            - O *default* é o caso onde o valor da variável não coincide com nenhum dos demais.
    - **Blocos de repetição**
        - Os dois blocos mais conhecidos são o **while** e o **for**.
        - **While**
            - Tradução literal: enquanto.

        - Funciona de forma similar ao **if**, ou seja, um conjunto de comandos/instruções é executado enquanto uma condição for verdadeira.
        - Ex.:
            - **while** condição
                - comandos
    - **For**
        - Esse bloco faz uma **iteração** sobre um conjunto de elementos, ou sobre uma faixa de valores. Ou seja, há uma "visita" a cada elemento/valor, ou a elementos/valores em intervalos específicos.
        - Ex.: considerando uma variável inteira x.
            - **for** x=1;x<10;x++;
            - comandos
        - No exemplo acima dissemos ao computador para ele começar a contar em 1, parar no número que for abaixo de nove, e continuar a contagem de 1 em 1.
        - Portanto, o computador vai fazer a seguinte contagem: 1, 2, 3, 4, 5, 6, 7, 8, 9. E para cada contagem, o conjunto de comandos/instruções é executado.
        - Existem variações do **for**, e o programador deve verificar na documentação da linguagem em quais situações se deve utilizar cada uma, e como utilizá-las.
- **Estruturas de dados**
    - Vamos ver, por enquanto, apenas as duas mais básicas: listas e Strings.
    - Uma lista é um conjunto de elementos. Como a lista é representada vai depender de cada linguagem de programação.
        - Uma lista simples é comumente chamada de vetor e *array*.
        - Ex.: lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    - Uma String, como já vimos antes, é um cadeia de caracteres, ou um conjunto de caracteres. Os caracteres podem estar organizados em uma lista.
        - Ex.: frase = [‘o’, ‘l’, ‘á’, ‘!’];