Revisão de Programação

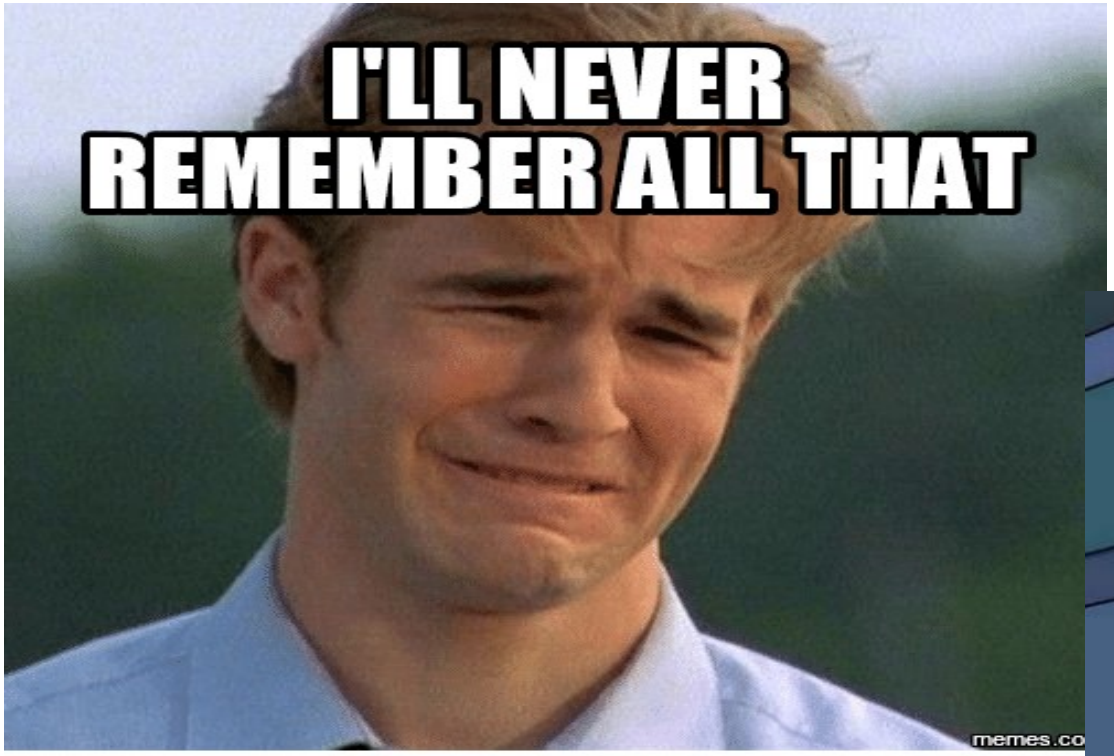
Pesquisa, Ordenação e Técnicas de Armazenamento

Prof. Msc. Bruno de A. Iizuka Moritani
bruno.moritani@anhembi.br

Agenda

- Conceitos necessários
  - Declaração de variáveis
  - Entrada e Saída de dados
  - Operadores lógicos
  - Estrutura de decisões
  - Estrutura de repetições
  - Vetores

Relembrando Lógica e Programação



Declaração de Variáveis

- Formato:
  - `<tipo> <nome da variável>;`
- Tipos mais usados:
  - int
  - double
  - float
  - boolean
  - char
  - String

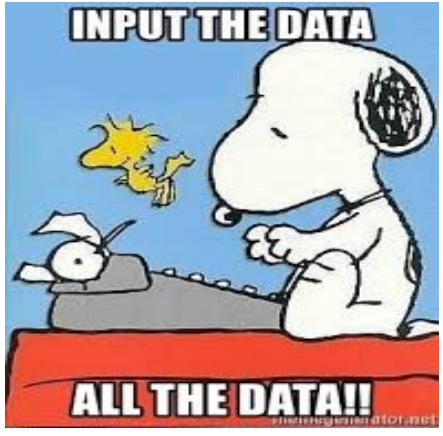
Declaração de Variáveis

- Exemplos:
  - `int idade = 21;`
  - `double precoGasolina = 4,499;`
  - `float peso = 85,5;`
  - `boolean casado = true;`
  - `char sexo = ‘f’;`
  - `String nome = “Bruno”;`

Saída de Dados

- Java:
  - `System.out.println(<mensagem>);`
    - <Mensagem>
      - Conjunto de caracteres
      - Variáveis
  - `System.out.println("Olá Mundo");`
  - `System.out.println(nome);`
  - `System.out.println(15+5);`

Entrada de Dados



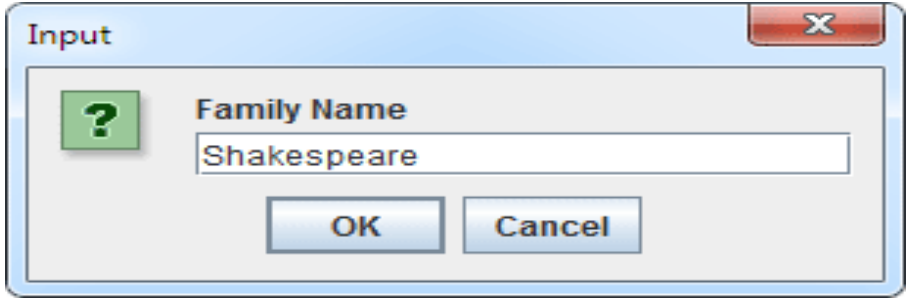
- Em Java:
  - Utiliza-se a classe `Scanner`
    - Presente no pacote `java.util`
      - Tem que importar a classe:
        - `import java.util.Scanner;`
    - Método mais fácil para se obter entradas de tipos primitivos:
      - `int`, `double`, `float`, `char`, `String`.

Entrada de Dados

- Para ler diferentes tipos da entrada em Java, usa-se diferentes comandos:
  - String:
    - `entrada.nextLine()`
  - int:
    - `entrada.nextInt()`
  - float:
    - `entrada.nextFloat()`
  - double:
    - `entrada.nextDouble()`
  - char:
    - `entrada.next().charAt(0)`
  - boolean:
    - `entrada.nextBoolean()`

Entrada e Saída de Dados

- Pode ser feita por meio da classe JOptionPane
  - JoptionPane.showMessageDialog(null, “String que irá ser exibida”);
  - JOptionPane.showInputDialog(“String do texto de entrada”);



Estrutura Básica - Java

```
1  public class NomeClasse{
2              public static void main (String args[]) {
3          //lógica aqui dentro
4      }
5  }
6
```

Operadores Lógicos

- E (&&)
  - Se duas expressões condicionais forem verdadeiras, o resultado é verdadeiro.
  - Tabela verdade:

A	B	A && B
V	V	V
V	F	F
F	V	F
F	F	F

Operadores Lógicos

- OU (||)
  - Se qualquer expressão condicional for verdadeira, o resultado é verdadeiro.
  - Tabela verdade:

A	B	A || B
V	V	V
V	F	V
F	V	V
F	F	F

Operadores Lógicos

- NÃO (!)
  - Se a expressão condicional for verdadeira, o resultado é falso.
  - Se a expressão condicional for falsa, o resultado é verdadeiro.
  - Tabela verdade:

A	!A
V	F
F	V

Precedência de Operadores

NÃO (!)

>

E (&&)

>

OU (||)

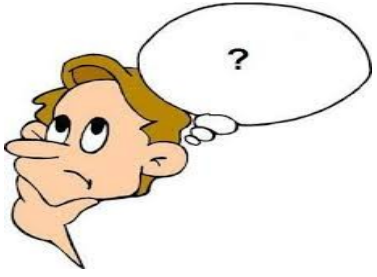
Condicional - `if`

- Habilidade de controlar o fluxo do seu programa;
- Permite a entrada em uma parte do código, dependendo da condição ser verdadeira ou falsa;

```
if (<expressão lógica>) {
   <instruções se expressão verdadeira> }
if (<expressão lógica>) {
   <instruções se expressão verdadeira>
} else {
   <instruções se expressão falsa>
}
```

Exercício Condicional - `if`

- Faça um programa em Java que receba:
  - o nome e a idade
- de duas pessoas e em seguida diga qual pessoa é mais velha.



Resposta Exercício - `if`

```java
import java.util.Scanner;

public class ExercicioConditionalIf {

    public static void main(String[] args) {

        String nome1, nome2;
        int idade1, idade2;

        //Primeira Pessoa
        Scanner entrada1 = new Scanner (System.in);
        System.out.print("Nome: ");
        nome1 = entrada1.nextLine();
        System.out.print("Idade: ");
        idade1 = entrada1.nextInt();

        //Segunda Pessoa
        Scanner entrada2 = new Scanner (System.in);
        System.out.print("Nome: ");
        nome2 = entrada2.nextLine();
        System.out.print("Idade: ");
        idade2 = entrada2.nextInt();

        //Comparações
        if (idade1 == idade2){
            System.out.println(nome1 + " e " + nome2 + " possuem a mesma idade");
        }else {
            if (idade1 > idade2){
                System.out.println(nome1 + " é mais velho(a) que " + nome2);

            } else{
                System.out.println(nome2 + " é mais velho(a) que " + nome1);
            }
        }

    }

}
```

Comparações entre Strings

- Quando se compara uma String com outra, não pode-se utilizar o comparador == igual
- Deve-se utilizar o método equals(<texto>).
  - Exemplo:
    - if (palavra == “Olá”) -> Errado
    - if (palavra.equals(“Olá”)) -> Correto

Repetições - `while`

```
while (<expressão lógica>) {
   <instruções enquanto expressão verdadeira>
}
```

- Usado preferencialmente quando não conhecemos o número de instruções a serem executadas.

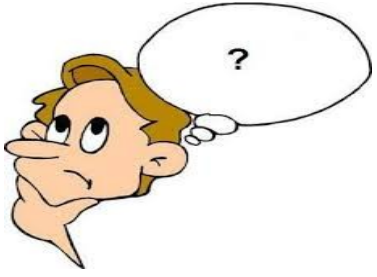
Repetições – do-while

```
do{
   <instruções enquanto expressão verdadeira>
}while (<expressão lógica>);
```

- Usado **quando precisamos ao menos executar uma vez** e não conhecemos o número de instruções a serem executadas.

Exercício Repetição

- Obtenha um número digitado pelo usuário e repita a operação de multiplicar ele por dois (imprimindo o novo valor) até que ele seja maior do que 100.
  - Ex.: se o usuário digita 8, deveremos observar na tela a seguinte sequência: 8 16 32 64 128
  - Usar `while` ou `do-while`



Resposta Exercício

```java
import java.util.Scanner;

public class Aula01ExercicioRepeticao {

    public static void main(String[] args) {
        int numero;

        Scanner entrada = new Scanner (System.in);
        System.out.print("Digite um número: ");
        numero = entrada.nextInt();
        System.out.print("\n" + numero + " ");

        while ( numero <= 100 ){
            numero = numero * 2;
            System.out.print(numero + " ");
        }
    }

}
```

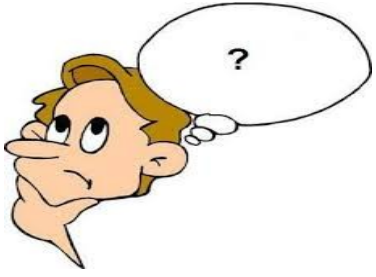
Repetições - `for`

```
for (<inicialização>; <expressão lógica>;
<atualiza a inicialização>){
   <instruções enquanto expressão verdadeira>
}
```

- Usado quando conhecemos a priori o número de instruções a serem executadas.

Exercício Repetição

- Leia um número inteiro,*n* , e mostre a somatória dos números que estão entre 1 e *n*



Resposta Exercício

```
package aula01exerciciorepeticaofor;

import java.util.Scanner;

public class Aula01ExercicioRepeticaoFor {

    public static void main(String[] args) {
        int n, somatoria = 0;

        Scanner entrada = new Scanner (System.in);
        System.out.print("Digite um número inteiro: ");
        n = entrada.nextInt();

        for (int i = 1; i <= n ; i++ ){
            somatoria = somatoria + i;
            System.out.println ("A somatória é igual a " + somatoria);
            System.out.println ("i " + i);
        }

        System.out.println ("A somatória é igual a " + somatoria);
    }

}
```

Vetores

- Declaração de vetores segue a forma:
  - `<tipo> <nome da variável> [] = **new** <tipo> [<tamanho do vetor>];`
- Para manipular os dados do vetor, usamos um índice que, começando em 0, diz qual a posição do vetor que queremos acessar.

Notas	10	8,5	4,5	7,5	6,5	5	8	9	9,5	8,5
Índice	0	1	2	3	4	5	6	7	8	9

Vetores

- double notas[]= new double[6];
- double valor;

- valor = notas[0];
- valor = notas[1];
- valor = notas[2];
- valor = notas[3];
- valor = notas[4];
- valor = notas[5];

Notas	10	8,5	4,5	7,5	6,5	5
Índice	0	1	2	3	4	5

Preenchendo um Vetor

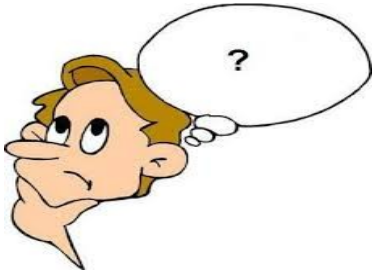
```
public class Program {
   public static void main(String args[]) {
      Scanner entrada = new Scanner(System.in);
      int Idades[] = new int[6];
      for(int i = 0; i < Idades.Length; i++) {
         Idades[i] = entrada.nextInt();
      }
   }
}
```

Escrevendo dados de um vetor

```
public class Program {
   public static void main(string[] args) {
      Scanner entrada = new Scanner(System.in);
      int Idades[] = new int[6];
      for (int i = 0; i < Idades.Length; i++)
         Idades[i] = entrada.nextInt();
      for (int i = 0; i < Idades.Length; i++)
         System.out.println(Idades[i]);
   }
}
```

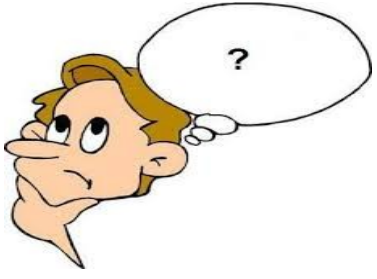
Exercícios

- Declare um vetor de 10 posições e o preencha com os 10 primeiros números impares e o escreva.
- Leia um vetor de 20 posições e em seguida um valor X qualquer. Seu programa devera fazer uma busca do valor de X no vetor lido e informar a posição em que foi encontrado ou se não foi encontrado.



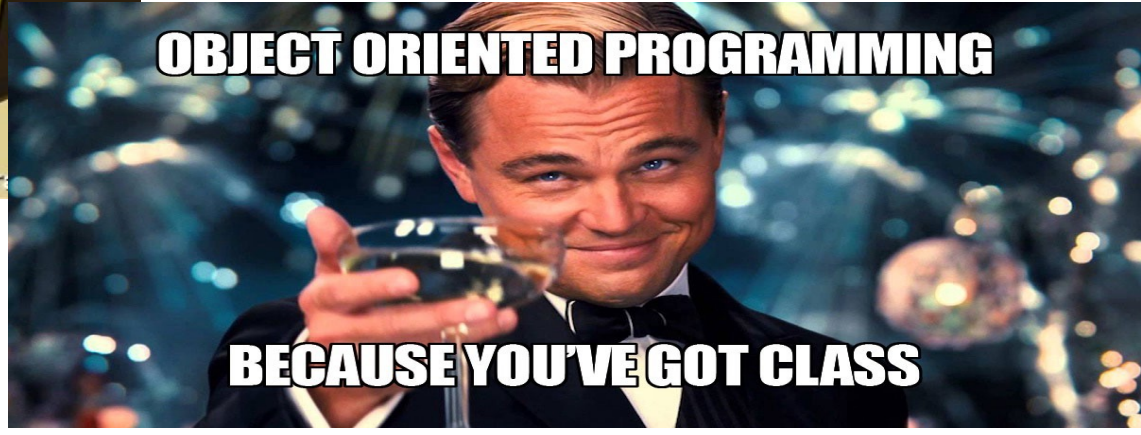
Exercícios

- Dadas as temperaturas que foram registradas diariamente durante uma semana, deseja-se determinar em quantos dias dessa semana a temperatura esteve acima da média. Escreva um programa que resolva esse problema e, após apresentar o resultado, pergunte ao usuário se ele deseja resolvê-lo novamente com outros dados.



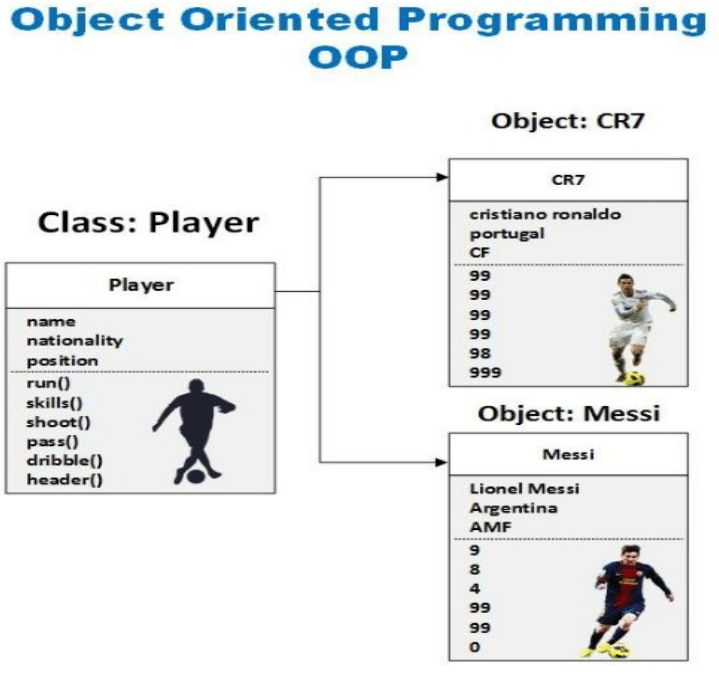

REMEMBER..
YOU'RE AWESOME

Revisão Orientação a Objeto



Classes

- Uma classe representa um grupo de elementos com:
  - Características (atributos)
  - Habilidades (métodos)
- em comum!



How to learn OOP using football

Classes

- Como podemos agrupar essas figuras?












Classes

Classes

- Características
  - Tipo de tela
  - Teclado
  - 4G
- Habilidades
  - Ligar
  - Enviar mensagem

Classes

- Características
  - Cor
  - Velocidade
  - Marcha
  - Motor
- Habilidades
  - Frear
  - Acelerar
  - Trocar Marcha

Definição de Classes

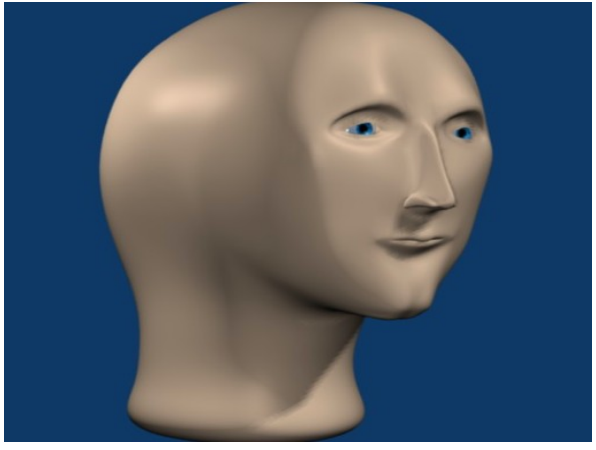
- Em Java, classes são definidas através do uso da palavra-chave **`class.`**
- Sintaxe para definir uma classe:

```
[modificador] class NomeDaClasse {
      // corpo da classe...
}
```

  - Após a palavra-chave `class`, segue-se o nome da classe, que deve ser um identificador válido para a linguagem.
  - `[modificador]` é opcional; se presente, pode ser uma combinação de **`public`** e **`abstract`** ou **`final.`**
    - O modificador **`abstract`** indica que nenhum objeto dessa classe pode ser instanciado.
    - O modificador **`final`** indica que a classe não pode ser uma superclasse (uma classe não pode herdar de uma classe final)

Exemplo

```
public class Pessoa {
   String nome;
   int idade;
}
```



Construtores

- Um construtor é um método especial, definido para cada classe.
  - Determina as ações associadas à inicialização de cada objeto criado.
  - É invocado toda vez que o programa instancia um objeto dessa classe.

Construtores

- A assinatura de um construtor diferencia-se das assinaturas dos outros métodos por não ter nenhum tipo de retorno (nem mesmo void).
- O nome do construtor deve ser o próprio nome da classe.
- O construtor pode receber argumentos, como qualquer método.
- Toda classe tem pelo menos um construtor sempre definido.

Exemplo - Construtores

```
public class Pessoa {
   private String nome;
   private int idade;
   public Pessoa(String nome, int idade) {
      this.nome = nome;
      this.idade = idade;
   }
   public Pessoa() {
      this(“Bruno”, 20);
   }
}
```

Instanciação

- A instanciação é um processo por meio do qual se realiza a cópia de um objeto (classe) existente.
- Uma classe, a qual tem a função de determinar um tipo de dado, deve ser instanciada para que possamos utilizá-la.
- Sendo assim, devemos criar sua instância, a qual definimos como sendo um objeto referente ao tipo de dado que foi definido pela classe.

Instanciação

- A criação do objeto é feita pelo operador **new**
  - `<Nome da Classe>  <nome do Objeto> = new <Nome da Classe>(<argumentos>);`
- Exemplos
  - `Pessoa pedro = new Pessoa("Pedro", 32);`
  - `Pessoa p1 = new Pessoa();`

Objetos

- Cada objeto se diferencia um do outro pelo valor de seus atributos

- Altura = 1.65
- Idade = 75
- Peso = 73

- Altura = 1.63
- Idade = 30
- Peso = 58

Referência `this`

- É uma referência a um objeto
- Quando um método de uma classe faz referência a outro membro dessa classe para um objeto específico dessa classe, como Java assegura que o objeto adequado recebe a referência?
  - Cada objeto tem uma referência a ele próprio - chamada de referência **`this`**
  - Utiliza-se a referência `this` implicitamente para fazer referências às variáveis de instância e aos métodos de um objeto

Referência `this`

- Exemplos de uso de `this`
  - A palavra-chave `this` é utilizada principalmente em dois contextos:
    - Diferenciar atributos de objetos, de parâmetros ou variáveis locais de mesmo nome;
    - Acessar o método construtor a partir de outros construtores.
- Utilizar `this` explicitamente pode aumentar a clareza do programa em alguns contextos em que `this` é opcional.

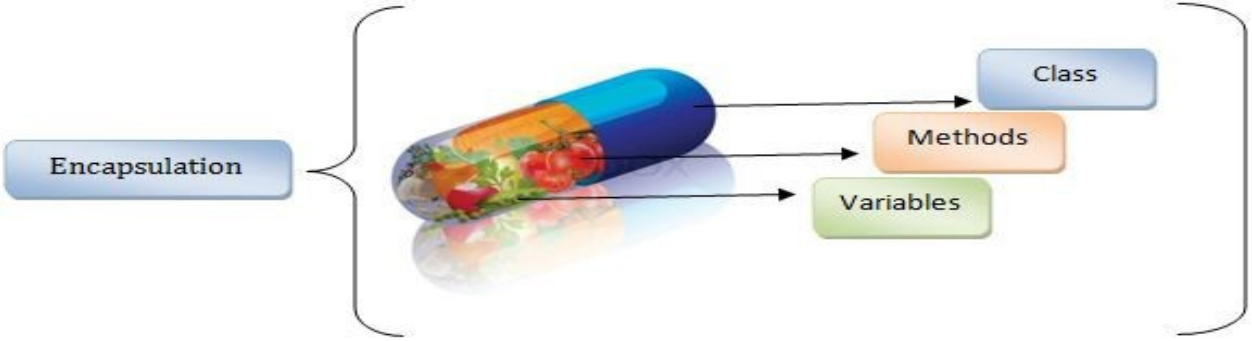
Referência `this`

- Esse exemplo ilustra esses dois usos:

```
public class EsteExemplo {
    int x;
    int y;
    // exemplo do primeiro caso:
    public EsteExemplo(int x, int y){
        this.x = x;
        this.y = y;
    }
    // exemplo do segundo caso:
    public EsteExemplo (){
        this(1, 1);
    }
}
```

Encapsulamento

- Permite com que os detalhes internos de funcionamento dos métodos permaneçam ocultos para os objetos
- Protege o acesso aos valores dos atributos
- Métodos “get” e “set”



Encapsulamento

- Public
  - Este é o modificador menos restritivo.
  - Métodos e atributos podem ser acessados pela sua classe e por todas as outras.
- Private
  - Este é o modificador mais restritivo e o mais comum.
  - Se utilizar este modificador com um atributo ou método, ele só pode ser acessado pela classe em que pertence. Sub-classes ou outras classes não pode acessar o atributo ou método declarado como private.
- Protected
  - Métodos e atributos podem ser acessados:
    - sua classe, classes do mesmo pacote e por suas sub-classes.
- Sem modificadores
  - Pode ser acessado pela sua classe e por todas as classes que estão no mesmo pacote.

Exercício Classe Automóvel

- Uma classe Automóvel com os seguintes atributos:
  - Nome do proprietário
  - Modelo
  - Placa
  - Ano
- É possível alterar o nome do proprietário e imprimir os dados do automóvel.
- Fazer uma classe que possibilite a transferência de proprietários.

Associação

- É um tipo de relacionamento entre classes
- Objetos de uma classe estão conectados a objetos de outra classe (ou da mesma classe)
- Representam relacionamento “tem um”
  - Livro “tem um” capítulo
  - Carro “tem uma” roda

Vetores de Objetos

- É possível criar um vetor para armazenar um conjunto de objetos de uma mesma classe
- Cada elemento do vetor representa um objeto desta classe

Vetores de Objetos

Pessoa	objeto	objeto	objeto	objeto	objeto	objeto	objeto
Índice	0	1	2	3	4	5	6

- Declaração de um vetor de objetos:
  - `Pessoa[] p = new Pessoa[6];`
- Acesso a um atributo:
  - `p[0].setNome(“Pedro”);`
  - `p[1].setNome(“Maria”);`

Dúvidas

Exercício Extra 01

- Neste problema, você deverá exibir uma lista de 1 a 100, um em cada linha, com as seguintes exceções:
  - Números divisíveis por 3 deve aparecer como 'Fizz' ao invés do número;
  - Números divisíveis por 5 devem aparecer como 'Buzz' ao invés do número;
  - Números divisíveis por 3 e 5 devem aparecer como 'FizzBuzz' ao invés do número.

Exercício Extra 02

- Jokenpo é uma brincadeira japonesa, onde dois jogadores escolhem um dentre três possíveis itens: Pedra, Papel ou Tesoura.
- O objetivo é fazer um juiz de Jokenpo que dada a jogada dos dois jogadores informa o resultado da partida.
- As regras são as seguintes:
  - Pedra empata com Pedra e ganha de Tesoura
  - Tesoura empata com Tesoura e ganha de Papel
  - Papel empata com Papel e ganha de Pedra

Exercício Extra 03

- Para definir uma sequência a partir de um número inteiro positivo, temos as seguintes regras:

$$n \rightarrow n/2 \text{ (n é par)}$$

$$n \rightarrow 3n + 1 \text{ (n é ímpar)}$$

- Usando a regra acima e iniciando com o número 13, geramos a seguinte sequência:

$$13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

- Podemos ver que esta sequência (iniciando em 13 e terminando em 1) contém 10 termos. Embora ainda não tenha sido provado (este problema é conhecido como Problema de Collatz), sabemos que com qualquer número que você começar, a sequência resultante chega no número 1 em algum momento. Elaborar um programa que dado um determinado número, informe a sequência de Collatz do mesmo.
- Extra: Desenvolva um programa que descubra qual o número inicial entre 1 e 1 milhão que produz a maior sequência.

Exercício Extra 04

- Um homem chamado Aristóteles é o responsável por ligar e desligar as luzes de um corredor. Cada lâmpada tem seu próprio interruptor que liga e a desliga. Inicialmente todas as lâmpadas estão desligadas.
- Aristóteles faz uma coisa peculiar: se existem n lâmpadas no corredor, ele caminha até o fim do corredor e volta n vezes. Na i-ésima caminhada, ele aperta apenas os interruptores aos quais sua posição é divisível por i. Ele não aperta nenhum interruptor na volta à sua posição inicial, apenas na ida. A i-ésima caminhada é definida como ir ao fim do corredor e voltar.
- Determine qual é o estado final de cada lâmpada. Está ligada ou desligada?
- Exemplo:
  - Entrada: 3
  - Saída: [on, off, off]

Obrigado

bruno.moritani@anhembi.br