–Biblioteca de Métodos–
P2

Nome: Humberto Corrêa Gomes

Data: 28/06/2022

Cálculo Das Áreas (1):

```
 public class CalculoAreas {
    public static double areaCirculo(double raio) {
        return Math.PI * Math.pow(raio, 2);
    }

    public static double areaRetangulo(double base, double altura) {
        return base * altura;
    }

    public static double areaQuadrado(double lado) {
        return Math.pow(lado, 2);
    }

    public static double areaTriangulo(double base, double altura) {
        return (base * altura) / 2;
    }
    public static void main(String args[]){
    System.out.println("Área do círculo: " + areaCirculo);
    System.out.println("Área do retângulo: " + areaRetangulo);
    System.out.println("Área do quadrado: " + areaQuadrado);
    System.out.println("Área do triângulo: " + areaTriangulo);
      }
}
```

Cálculos das áreas laterais (2):

```
public class CalculoAreasLaterais{

    public static double areaEsfera(double raio) {

        return 4 * Math.PI * Math.pow(raio, 2);

    }

    public static double areaLateralCilindro(double altura, double raio) {

        return 2 * Math.PI * raio * altura;

    }

    public static double areaLateralPrismaTriangular(double base, double
altura, double lateral) {

        return (base + lateral + lateral) * altura;

    }

    public static double areaLateralPrismaQuadrada(double ladoBase, double
altura) {

        return 4 * ladoBase * altura;

    }

    public static double areaLateralPrismaRetangular(double larguraBase,
double profundidadeBase, double altura) {

        return 2 * altura * (larguraBase + profundidadeBase);

    }

    public static double areaLateralPiramideTriangular(double base, double
altura) {

        double lado = Math.sqrt(Math.pow(base / 2, 2) + Math.pow(altura, 2));

        return 3 * lado * altura / 2;

    }

    public static double areaLateralPiramideQuadrada(double ladoBase, double
altura) {

        double apotema = Math.sqrt(Math.pow(ladoBase / 2, 2) +
Math.pow(altura, 2));

        return 4 * apotema * ladoBase / 2;

    }

    public static double areaLateralPiramideRetangular(double larguraBase,
double profundidadeBase, double altura) {

        double lado = Math.sqrt(Math.pow(larguraBase / 2, 2) +
Math.pow(altura, 2));

        double lado2 = Math.sqrt(Math.pow(profundidadeBase / 2, 2) +
Math.pow(altura, 2));

        return 2 * (lado * larguraBase + lado2 * profundidadeBase);

    }}
```

Calculo Volume (3)

```
public class CalculoVolume {
    public static double volumeEsfera(double raio) {
        return (4.0 / 3.0) * Math.PI * Math.pow(raio, 3);
    }
    public static double volumeCilindro(double raio, double altura) {
        return Math.PI * Math.pow(raio, 2) * altura;
    }
    public static double volumeCubo(double aresta) {
        return Math.pow(aresta, 3);
    }
    public static double volumePrismaTriangular(double base, double altura,
double comprimento) {
        return (base * altura * comprimento) / 2;
    }
    public static double volumePrismaQuadrada(double ladoBase, double altura)
{
        return Math.pow(ladoBase, 2) * altura;
    }
    public static double volumePrismaRetangular(double larguraBase, double
comprimentoBase, double altura) {
        return larguraBase * comprimentoBase * altura;
    }
}
```

Contar Divisores (4):

```
public class ContarDivisores{
    public static int ContadorD(int numero) {
        int contador = 0;
        for (int i = 1; i <= numero; i++) {
            if (numero % i == 0) {
                contador++;
            }
        }
        return contador;
    }
}
```

Divisores próprios (5)

```
public class DivisoresProprios {
    public static int qtdDivisoresProprios(int n) {
        int qtd = 0;
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                if (i * i == n) {
                    qtd += 1;
                } else {
                    qtd += 2;
                }
            }
        }
        return qtd;
    }
}
```

Cálculos de figuras geométricas (6):

```
public class FigurasGeometricas {
    public static double calcularAreaCirculo(double raio) {
        double pi = 3.14;
        return pi * raio * raio;
    }
    public static double calcularAreaRetangulo(double base, double altura) {
        return base * altura;
    }
    public static double calcularAreaQuadrado(double lado) {
        return lado * lado;
    }
    public static double calcularAreaTriangulo(double base, double altura) {
        return (base * altura) / 2;
    }
    public static double calcularPerimetroCirculo(double raio) {
        double pi = 3.14;
        return 2 * pi * raio;
    }
calcularPerimetroRetangulo(double base, double altura) {
        return 2 * (base + altura);
    }
    public static double calcularPerimetroQuadrado(double lado) {
        return 4 * lado;
    }
    public static double calcularPerimetroTriangulo(double lado1, double
lado2, double lado3) {
        return lado1 + lado2 + lado3;
    }
}
```

Leitura de 3 Números (7)

```
public class Ler3 {
    public static int[] ordemDecrescente(int num1, int num2, int num3){
        int[] numeros = { num1, num2, num3 };
        Arrays.sort(numeros);
        for (int i = 0; i < numeros.length / 2; i++) {
            int temp = numeros[i];
            numeros[i] = numeros[numeros.length - 1 - i];
            numeros[numeros.length - 1 - i] = temp;
        }
        return numeros;
    }
    public static void main(String args[]){
        int num1, num2, num3;
        try (Scanner entrada = new Scanner(System.in)) {
            System.out.println("Digite o primeiro número");
            num1 = entrada.nextInt();
            do {
                System.out.println("Digite o segundo número");
                num2 = entrada.nextInt();
                if(num2==num1) {
                    System.out.println("Os números não podem ser iguais");
                }
            } while(num2==num1);
            do {
                System.out.println("Digite o terceiro número");
                num3 = entrada.nextInt();
                if(num3==num1 || num3==num2) {
                    System.out.println("Os números não podem ser iguais");
                }
            } while(num3==num1 || num3==num2);

            int[] numeros = ordemDecrescente(num1, num2, num3);
            System.out.println(numeros[0] + ", " + numeros[1] + ", " + numeros[2]);
        }
    }}
```

Número Abundante (8):

```
public class NumAbundante {
    public static boolean Abundante(int num) {
        int sum = 0;
        for (int i = 1; i <= num/2; i++) {
            if (num % i == 0) {
                sum += i;
            }
        }
        return sum > num;
    }
}
```

Números Amigos (9):

```
public class NumAmigos {
    public static boolean Amigos(int num1, int num2) {
        int soma1 = 0, soma2 = 0;
        for(int i = 1; i < num1; i++) {
            if(num1 % i == 0) {
                soma1 += i;
            }
        }
        for(int i = 1; i < num2; i++) {
            if(num2 % i == 0) {
                soma2 += i;
            }
        }
        if(soma1 == num2 && soma2 == num1) {
            return true;
        } else {
            return false;
        }
    }
}
```

Número Defectivo (10):

```
public class NumDefectivo {
    public static boolean Defectivo(int number) {
        int sum = 0;
        for (int i = 1; i <= number/2; i++) {
            if (number % i == 0) {
                sum += i;
            }
        }
        return (sum < number);
    }
}
```

Número Perfeito (11):

```
public class NumPerfeito {
    public static boolean NPerfeito(int num) {
        int somaDivisores = 0;
        for (int i = 1; i <= num / 2; i++) {
            if (num % i == 0) {
                somaDivisores += i;
            }
        }
        return somaDivisores == num;
    }
}
```

Soma de divisores (12)

```
public class SomaDivisores {
    public static int somaDivisores(int numero) {
        int soma = 0;
        for (int i = 1; i <= numero; i++) {
            if (numero % i == 0) {
                soma += i;
            }
        }
        return soma;
    }
}
```

Soma de divisores próprios (13)

```
public class SomaDivisoresP {
    public static int somaDivisoresProprios(int n) {
        int soma = 0;
        for (int i = 1; i <= n/2; i++) {
            if (n % i == 0) {
                soma += i;
            }
        }
        return soma;
    }
}
```

Verificar Primo (14):

```
public class VerifPrimo {
    public static boolean Primo(int n) {
        if (n <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Trocar posição do par para ímpar (15)

```
import java.util.Arrays;
```

```java
public class TrocarParImpar{
    public static void trocarPosicaoParImpar(int[] vetor) {
        int tamanho = vetor.length;
        int[] resultado = new int[tamanho];
        int indicePar = 0;
        int indiceImpar = tamanho - 1;
        for (int i = 0; i < tamanho; i++) {
            if (vetor[i] % 2 == 0) {
                resultado[indicePar] = vetor[i];
                indicePar++;
            } else {
                resultado[indiceImpar] = vetor[i];
                indiceImpar--;
            }
        }
        for (int i = indicePar; i <= indiceImpar; i++) {
            resultado[i] = vetor[i];
        }
        System.arraycopy(resultado, 0, vetor, 0, tamanho);
    }
    public static void main(String[] args) {
        int[] vetor = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        System.out.println("Vetor original: " + Arrays.toString(vetor));
        trocarPosicaoParImpar(vetor);
        System.out.println("Vetor modificado: " + Arrays.toString(vetor));
    }
}
```

Explicações:

Cálculo da Área (1):

O método "`areaCirculo`" recebe o raio como parâmetro e utiliza a fórmula da área do círculo ($\pi * r^2$) para calcular a área e retorná-la.

O método "`areaRetangulo`" recebe a base e a altura como parâmetros e utiliza a fórmula da área do retângulo ($base * altura$) para calcular a área e retorná-la.

O método "`areaQuadrado`" recebe o lado como parâmetro e utiliza a fórmula da área do quadrado ($lado^2$) para calcular a área e retorná-la.

O método "`areaTriangulo`" recebe a base e a altura como parâmetros e utiliza a fórmula da área do triângulo ($\frac{(base*altura)}{2}$) para calcular a área e retorná-la.

Cálculo da Área Lateral (2):
O código acima define uma classe Java chamada `"CalculoAreasLaterais"`, que contém métodos estáticos para calcular a área lateral de vários sólidos geométricos.

Os métodos da classe incluem:

- `"areaEsfera"`: calcula a área da superfície de uma esfera, dado o raio.
- `"areaLateralCilindro"`: calcula a área lateral de um cilindro, dado a altura e o raio da base.
- `"areaLateralPrismaTriangular"`: calcula a área lateral de um prisma de base triangular, dado a base, a altura e um dos lados laterais.
- `"areaLateralPrismaQuadrada"`: calcula a área lateral de um prisma de base quadrada, dado o comprimento do lado da base e a altura.
- `"areaLateralPrismaRetangular"`: calcula a área lateral de um prisma de base retangular, dado a largura da base, a profundidade da base e a altura.
- `"areaLateralPiramideTriangular"`: calcula a área lateral de uma pirâmide de base triangular, dado a base e a altura.
- `"areaLateralPiramideQuadrada"`: calcula a área lateral de uma pirâmide de base quadrada, dado o comprimento do lado da base e a altura.
- `"areaLateralPiramideRetangular"`: calcula a área lateral de uma pirâmide de base retangular, dado a largura da base, a profundidade da base e a altura.

Cada método usa fórmulas matemáticas diferentes para calcular a área lateral do sólido geométrico correspondente, e retorna o resultado como um valor de ponto flutuante `(double)`. Todos os métodos são definidos como estáticos, o que significa que podem ser chamados diretamente na classe, sem a necessidade de criar uma instância dela.

Cálculo Volume (3):
O código define uma classe chamada `"CalculoVolume"` que contém vários métodos

para calcular o volume de diferentes sólidos geométricos. Cada método recebe parâmetros específicos, dependendo do sólido que se deseja calcular o volume.

Os métodos incluem:

- `"volumeEsfera"`: recebe o raio da esfera e retorna o volume dela.
- `"volumeCilindro"`: recebe o raio e a altura do cilindro e retorna o volume dele.
- `"volumeCubo"`: recebe a aresta do cubo e retorna o volume dele.
- `"volumePrismaTriangular"`: recebe a base, altura e comprimento do prisma de base triangular e retorna o volume do mesmo.
- `"volumePrismaQuadrada"`: recebe o lado da base e a altura do prisma de base quadrada e retorna o volume dele.
- `"volumePrismaRetangular"`: recebe a largura e o comprimento da base, além da altura do prisma de base retangular e retorna o volume dele.

Todos os cálculos são feitos utilizando fórmulas matemáticas específicas para cada sólido geométrico.

Contar Divisores (4):

Esse código implementa uma função chamada **"`ContadorD`"** que recebe um número inteiro como argumento e conta quantos divisores esse número tem.

A função começa inicializando um contador com o valor zero. Em seguida, um "`loop for`" é usado para iterar por todos os números de 1 até o número fornecido como argumento. Dentro do "`loop`", o número é verificado se é divisível pelo iterador atual usando o operador `%` (resto da divisão). Se o resultado for 0, isso significa que o número é um divisor do número fornecido e o contador é incrementado.

Divisores próprios (5):

Esse código implementa um método chamado **"`qtdDivisoresProprios`"** que recebe um número inteiro "$\boldsymbol{N}$" como parâmetro e retorna à quantidade de divisores próprios de "$\boldsymbol{N}$".

Os divisores próprios de um número "$\boldsymbol{N}$" são todos os divisores de "$\boldsymbol{N}$", exceto por ele próprio. Por exemplo, os divisores próprios de 12 são $1, 2, 3, 4$ e 6.

O método começa inicializando uma variável "$\boldsymbol{qtd}$" com o valor zero. Em seguida, um loop for é utilizado para percorrer todos os números inteiros de 2 até a raiz quadrada de $\boldsymbol{N}$ (o valor máximo possível para um divisor de $\boldsymbol{N}$).

Dentro do "`loop`", o método verifica se **"$\boldsymbol{N}$"** é divisível por "$\boldsymbol{I}$" (usando a operação módulo %). Se for, isso significa que "$\boldsymbol{I}$" é um divisor de "$\boldsymbol{N}$". O método então verifica se "$\boldsymbol{I}$" é um divisor próprio de "$\boldsymbol{N}$", isto é, se "$\boldsymbol{I}$" é diferente de "$\boldsymbol{N}$". Se "$\boldsymbol{I}$" for igual a "$\boldsymbol{N}$", isso significa que "$\boldsymbol{N}$" não tem divisores próprios, então o método simplesmente retorna 0.

Se "$\boldsymbol{I}$" for um divisor próprio de "$\boldsymbol{N}$", o método incrementa a variável "$\boldsymbol{qtd}$". Em seguida, o método verifica se o quadrado de "$\boldsymbol{I}$" é igual a "$\boldsymbol{N}$". Se for, isso significa que "$\boldsymbol{I}$" é a raiz quadrada de "$\boldsymbol{N}$" e não deve ser contado duas vezes. Nesse caso, o método incrementa "$\boldsymbol{qtd}$" em 1. Caso contrário, "$\boldsymbol{I}$" não é a raiz quadrada de "$\boldsymbol{N}$" e deve ser contado duas vezes (uma vez como divisor de "$\boldsymbol{N}$" e outra vez como divisor próprio). Nesse caso, o método incrementa "$\boldsymbol{qtd}$" em 2.

Após o "`loop`", o método retorna o valor de "$\boldsymbol{qtd}$", que representa a quantidade de divisores próprios de "$\boldsymbol{N}$".

Cálculo da área de figuras geométricas (6):

Esse código implementa um método chamado "`FigurasGeometricas`" que é uma biblioteca de funções para cálculo de áreas e perímetros de figuras geométricas básicas como círculo, retângulo, quadrado e triângulo.

Os métodos contidos na classe **"`FigurasGeometricas`"** são:

"`calcularAreaCirculo`**"**: recebe como argumento o raio do círculo e retorna a área correspondente.

"`calcularAreaRetangulo`": recebe como argumento a base e a altura do retângulo e retorna a área correspondente.

"`calcularAreaQuadrado`": recebe como argumento o lado do quadrado e retorna a área correspondente.

"`calcularAreaTriangulo`": recebe como argumento a base e a altura do triângulo e retorna a área correspondente.

"`calcularPerimetroCirculo`": recebe como argumento o raio do círculo e retorna o perímetro correspondente.

"`calcularPerimetroRetangulo`": recebe como argumento a base e a altura do retângulo e retorna o perímetro correspondente.

"`calcularPerimetroQuadrado`": recebe como argumento o lado do quadrado e retorna o perímetro correspondente.

"`calcularPerimetroTriangulo`": recebe como argumento os três lados do triângulo e retorna o perímetro correspondente.

Os cálculos de áreas e perímetros são feitos com base em fórmulas básicas da geometria, utilizando as informações passadas como argumentos para os métodos.

Ler 3 números (7):

Criamos três variáveis para armazenar os números digitados pelo usuário: "`num1`, `num2`, `num3`".

Pedimos ao usuário que digite “num1” e, em seguida, usamos um “loop do-while” para pedir ao usuário que digite “num2” até que “num2” seja diferente de “num1”. Em seguida, usamos outro “loop do-while” para pedir ao usuário que digite “num3” até que “num3” seja diferente de “num1” e “num2”.

Criamos três variáveis temporárias: “maior, meio, menor”. Inicializamos essas variáveis com os valores de “num1, num2, num3”, respectivamente. Usando condicionais “if”, verificamos qual é o maior número digitado e colocamos esse valor na variável “maior”. Em seguida, verificamos qual é o menor número digitado e colocamos esse valor na variável “menor”. Por fim, colocamos o número restante na variável “meio”. Imprimimos os valores de “maior, meio, menor” em ordem decrescente.

Número Abundante (8):

O código acima apresenta uma classe chamada "NumAbundante" com um método estático "Abundante", que recebe um número inteiro como argumento e retorna um valor “booleano” indicando se o número é abundante ou não.

A definição de um número abundante é um número inteiro positivo que é menor do que a soma de seus divisores próprios. Os divisores próprios de um número são os divisores estritamente menores do que o próprio número.

O método "Abundante" calcula a soma dos divisores próprios do número fornecido como entrada, verificando se cada um dos números inteiros menores que o número fornecido divide o número fornecido sem deixar um resto. Caso isso ocorra, a soma desses divisores é acumulada em uma variável chamada "sum". Em seguida, o método retorna “true” se a soma dos divisores próprios for maior do que o número original, e “false” caso contrário.

Por exemplo, se o número de entrada for 12, os divisores próprios de 12 são $1, 2, 3, 4, 6$. A soma desses divisores é 16, que é maior que 12. Portanto, o método "Abundante" retorna verdadeiro para 12.

Números Amigos (9):

Esse método recebe dois números inteiros como parâmetros e calcula a soma dos divisores próprios de cada um deles. Depois, verifica se a soma dos divisores próprios de um número é igual ao outro número e vice-versa. Se for verdadeiro, retorna “true”, indicando que os números são amigos. Caso contrário, retorna “false”.

Número Defectivo (10):

O método “Defectivo” recebe como parâmetro um número inteiro “number” e percorre todos os números inteiros positivos menores que “number” para verificar se são divisores de “number”. A soma de todos os divisores de “number” é armazenada na

variável “sum”. Se a soma dos divisores for menor que “number”, o número é considerado defectivo e o método retorna “true”. Caso contrário, o número não é defectivo e o método retorna “false”.

Número Perfeito (11):

O código acima é uma classe Java chamada “NumPerfeito” que contém um único método público chamado “NPerfeito”. Esse método recebe um número inteiro como parâmetro e retorna um valor “booleano” que indica se o número é ou não um número perfeito.

Um número perfeito é um número inteiro positivo que é igual à soma de seus divisores próprios (divisores diferentes de si mesmo). Por exemplo, o número 6 é um número perfeito, porque os seus divisores próprios são $1, 2, 3,$ e $1 + 2 + 3 = 6$.

O método “NPerfeito” começa inicializando uma variável inteira chamada “somaDivisores” com o valor 0. Em seguida, ele entra em um “loop for” que começa em 1 e termina em $\frac{num}{2}$ (já que não é necessário verificar divisores acima da metade do número). Dentro do “loop”, o método verifica se o número é divisível por “I”, usando o operador módulo (%). Se for, o valor de “I” é adicionado à variável “somaDivisores”.

Por fim, o método retorna um valor “booleano” que indica se a soma de todos os divisores próprios é igual ao número original (ou seja, se o número é ou não um número perfeito). Se a soma dos divisores próprios for igual ao número original, o método retorna “true”. Caso contrário, ele retorna “false”.

Soma de Divisores (12):

O código acima define uma classe chamada "SomaDivisores" com um método estático chamado "somaDivisores". O objetivo deste método é calcular a soma de todos os divisores de um número inteiro passado como argumento.

O método começa inicializando uma variável "soma" como 0. Em seguida, um “loop for” é usado para iterar sobre todos os inteiros de 1 a "numero". Para cada inteiro "I" no “loop”, o método verifica se "numero" é divisível por "I". Se sim, "I" é um divisor de "numero" e é adicionado à variável "soma". No final do loop, a variável "soma" contém a soma de todos os divisores de "numero".

O método retorna a variável "soma" como resultado da soma dos divisores do número.

Soma de divisores próprios (13):

Esse código implementa um método chamado “**somaDivisoresProprios**” que recebe um número inteiro “N” como parâmetro e retorna a soma dos divisores próprios desse número.

A variável "`soma`" é inicializada com 0 e, em seguida, um "`loop for`" é executado, começando com "$\boldsymbol{I = 1}$" e indo até "$\boldsymbol{I}$" ser menor ou igual a $\frac{\boldsymbol{n}}{\boldsymbol{2}}$. A cada iteração do "`loop`", verifica-se se "$\boldsymbol{N}$" é divisível por "$\boldsymbol{I}$" (ou seja, "$\boldsymbol{N \% I == 0}$"). Se for, "$\boldsymbol{I}$" é um divisor próprio de "$\boldsymbol{N}$" (pois "$\boldsymbol{I}$" é menor que "$\boldsymbol{N}$" e não inclui "$\boldsymbol{N}$"), e é adicionado à variável "`soma`".

Após o "`loop`", o método retorna o valor de "`soma`", que é a soma dos divisores próprios de "$\boldsymbol{N}$".

Verificar Número primo (14):

O código acima é uma implementação do método para verificar se um número inteiro é primo ou não. O método recebe um número inteiro como parâmetro e retorna um valor "`booleano`" que indica se o número é primo ou não.

Primeiro, o método verifica se o número é menor ou igual a 1, pois os números menores ou iguais a 1 não são considerados primos. Caso o número seja menor ou igual a 1, o método retorna "**`false`**".

Caso o número seja maior que 1, o método utiliza um "**`loop for`**" para verificar se há algum divisor do número que não seja 1 ou ele mesmo. O "**`loop`**" é executado de 2 até a raiz quadrada do número, pois não é necessário verificar os divisores maiores que a raiz quadrada, já que os outros divisores seriam repetidos. Se o número tiver um divisor que não seja 1 ou ele mesmo, o método retorna "**`false`**", caso contrário, o número é considerado primo e o método retorna "**`true`**".

Trocar posição do par para ímpar (15):

O método **`trocarPosicaoParImpar`** recebe um vetor de inteiros como entrada e realiza a troca de posição dos números pares pelos ímpares dentro do vetor. A lógica é percorrer o vetor, e se o número for par, ele é colocado no início do vetor resultado (incrementando o índice **`indicePar`**), caso contrário, o número é colocado no final do vetor resultado (decrementando o índice **`indiceImpar`**). Em seguida, os elementos restantes (se houver) são copiados no final do vetor resultado, e finalmente, o vetor resultado é copiado de volta para o vetor original usando o método "**`System.arraycopy`**"