

# **Programação de computadores II**

## **Trabalho 1: Calculadora básica e de raízes de polinômios de grau 2**

**Carolina Santos Lins<sup>1</sup>, Gustavo Correa Rodrigues<sup>1</sup>,  
Livia Emanuelle Carrera Soares da Silva<sup>1</sup>, Thiago Calado Sosinho<sup>1</sup>**

<sup>1</sup>Faculdade de Computação – Instituto de Ciências Exatas e Naturais  
Universidade Federal do Pará (UFPA)  
Av. Augusto Correa 01, 66075-090 – Belém – PA – Brasil

{carolina.lins, gustavo.rodrigues, livia.silva, thiago.sosinho}@icen.ufpa.br

### **1. Introdução**

O relatório tem como objetivo o desenvolve sistema de recursos humanos para uma empresa fictícia de tecnologia.. Estes programas foram construídos na linguagem de programação Java e utilizados os conceitos presentes no paradigma de orientação a objetos. A calculadora básica deve, contém as seguintes operações matemáticas: soma, subtração, multiplicação, divisão, exponenciação e resto da divisão.

As operações matemáticas são binárias, as quais recebem dois parâmetros por argumento e retornam o resultado da operação, além disso, a calculadora de raízes reais de polinômios de 2º grau e resolvendo a equação  $ax^2 + bx + c = 0$ , encontrando as raízes reais se caso existirem.

Tem-se um menu de usuário para utilização da calculadora, nele abrangendo duas opções principais, sendo a primeira ao entrar com a opção número 1 ir para a calculadora java possuidora das operações matemáticas básicas e, na segunda ao entrar com a opção número 2, ir para a calculadora de raízes, também há a opção número 0 para o termino do programa em questão. O menu estabelece um loop até que enfim o usuário escolha a opção 0 e encerre o programa.

### **2. Modelagem do Software**

Para estruturar as informações de forma que arquitetos e programadores possam compreendê-las, é necessário a intermediação do processo com diferentes opções de modelagem. Entretanto, a técnicas a ser utilizada neste trabalho trata-se da Modelagem de Software com UML (Unified Modeling Language), a linguagem mais utilizada quanto ao assunto. Uma das práticas comumente defendidas é a de que modelar uma solução técnica preferencialmente utilizando UML melhora a qualidade da solução e a manutenibilidade de um produto de software.

Com seus 14 diferentes diagramas (na versão 2.5), a UML defini-se de forma simples como um conjunto de diagramas que representam elementos, características e comportamentos de um software, essa predileção compreende para o programador como ele deve modificar os requisitos do software em código, ao mesmo tempo em que respeita a arquitetura proposta, contudo, os diagramas escolhidos para tal representação são: Diagramas de Casos de Uso, Diagrama de classes e Diagrama de Atividades.

## 2.1. Diagrama de Casos de Uso

Esse diagrama representa um conjunto sequenciado de ações que um sistema executa para gerar um resultado de valor observável por um ator. Os casos de uso também podem ser organizados pela inclusão, especificação de relacionamentos de generalização e extensão, existentes entre eles. Aplicam-se esses relacionamentos com a finalidade de partilhar o comportamento comum, obtendo esse comportamento a partir de outros casos de uso que ele inclui, e fatorar variantes, obtendo esse comportamento de outros casos de uso que o estendem. Organizar os casos de uso, extraindo o comportamento comum (por meio de inclusão) e diferenciando as variantes (por relacionamentos estendidos), é uma parte importante para a criação de um conjunto de casos de uso equilibrado e compreensível.

A modelagem dos diagramas deve ser direcionada de acordo os objetivos da utilização dos casos de uso, sendo os objetivos principais:

- Delimitação do contexto de um sistema;
- Documentação e o entendimento dos requisitos;
- Descrição dos requisitos funcionais;
- Principal saída da etapa de especificação de requisitos;
- Principal entrada da etapa de análise.

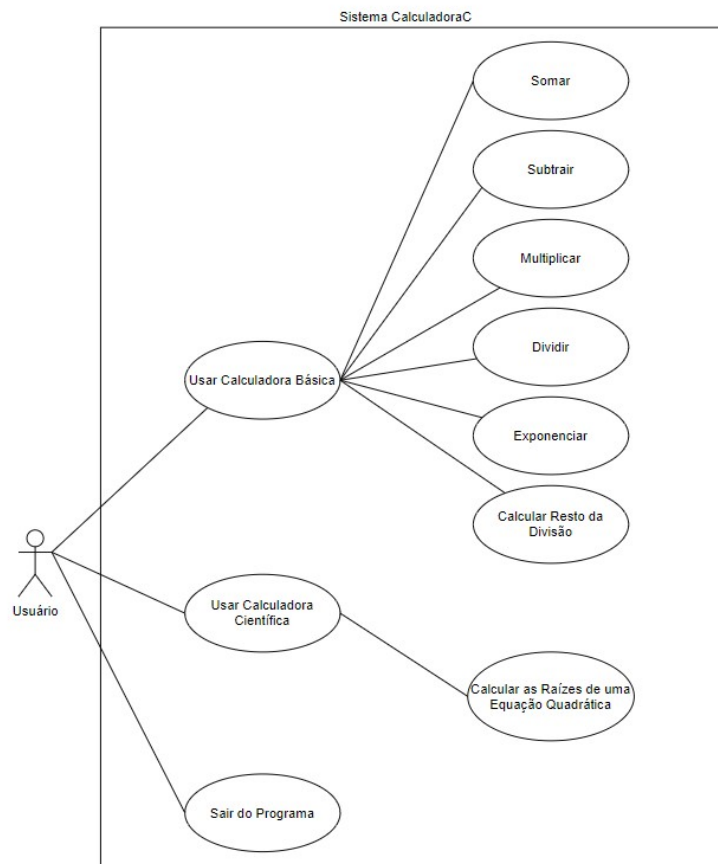
Mas também há os objetivos secundários, que são:

- Facilitar a comunicação entre os stakeholders;
- Servir de base para a definição do cronograma;
- Auxiliar na elaboração dos casos de teste;
- Possibilitar definição de uma estrutura analítica do sistema.

Já as características dos Diagramas de Casos de Uso são importantes dentro do contexto de especificação de documentação de um sistema, sendo as principais:

- Mostram um conjunto de casos de uso, atores e seus relacionamentos;
- Modelam aspectos dinâmicos do sistema;
- Proporcionam uma representação contextual do sistema (fronteira explícita);
- Indicam a forma como o sistema interage com as entidades externas (atores).

Logo, tendo em vista esse breve resumo sobre seus objetivos e característica foi formulado o diagrama a seguir com a representação do Sistema CalculadoraC.



**Figura 1. Sistema CalculadoraC, diagrama de Caso de Uso**

No diagrama da Figura 14 há todas as opções que o usuário pode escolher ao utilizar a calculadora.

Apesar de mostrado no diagrama, não houve necessidade de implementar uma classe Usuário.

## 2.2. Diagrama de Classes

Neste diagrama há mapeamento de forma clara da estrutura de um sistema ao modelar suas classes, seus atributos, operações e relações entre objetos. Estando entre os tipos mais úteis de diagramas UML, é um tipo de diagrama da estrutura porque descrevem o que deve estar presente no sistema a ser modelado. A UML é como um modelo padronizado para descrever uma abordagem de programação orientada ao objeto, sendo assim como as classes os componentes básicos dos objetos, diagramas de classes são os componentes básicos da UML.

Os diversos componentes em um diagrama de classes podem representar as classes que serão realmente programadas, os principais objetos ou as interações entre classes e objetos. A Classe em si consiste em um retângulo com três linhas, sendo a linha superior o nome da classe, a linha do meio os atributos da classe e a linha inferior os métodos ou operações que a classe pode utilizar. Classes e subclasses são agrupadas juntas para mostrar a relação estática entre cada objeto.

Como benefícios de diagramas de classes, tem-se:

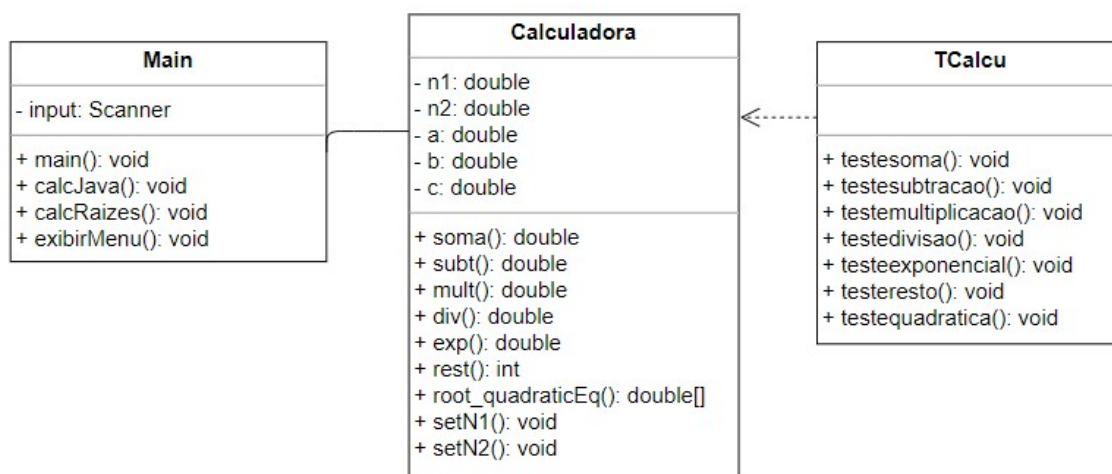
- Ilustrar modelos de dados para sistemas de informação, não importa quão simples ou complexo entendendo melhor a visão geral dos esquemas de uma aplicação;
- Expressar visualmente as necessidades específicas de um sistema e divulgar essas informações por toda a empresa;
- Criar gráficos detalhados que destacam qualquer código específico necessário para ser programado e implementado na estrutura descrita;
- Fornecer uma descrição independente de implementação de tipos utilizados em um sistema e passados posteriormente entre seus componentes;

O diagrama de classes padrão é composto de três partes:

- Parte superior: contém o nome da classe. Esta parte é sempre necessária, seja falando do classificador ou de um objeto;
- Parte do meio: contém os atributos da classe. Use esta parte para descrever as qualidades da classe. É necessário somente quando se descreve uma instância específica de uma classe;
- Parte inferior: inclui as operações da classe (métodos). Exibido em formato de lista, cada operação ocupa sua própria linha. As operações descrevem como uma classe interage com dados.

Ademais, uma Classe representa um objeto ou um conjunto de objetos que compartilham uma estrutura e comportamento comum. São representadas por um retângulo que inclui linhas do nome da classe, seus atributos e suas operações.

Novamente com este resumo sobre suas característica foi formulado o diagrama a seguir com a representação do Sistema CalculadoraC.



**Figura 2. Sistema CalculadoraC, diagrama de Classe**

Há três classes nesse sistema: Main, Calculadora e TCalcu.

Na classe Main, foram implementadas as funções que irão mostrar as opções e as instruções para o Usuário.

Na classe TCalcu há funções para testar cada funcionalidade da Calculadora. Detalhes de como essa classe funciona serão discutidos no tópico 3. Testes Computacionais.

Na classe Calculadora, há a implementação de todas as funções que farão as operações matemáticas da calculadora básica e da calculadora de polinômios. Os atributos n1 e n2 representam os números que serão utilizados nas operações binárias (soma, subtração, multiplicação, divisão, exponenciação e resto da divisão) e os atributos a, b e c representam os coeficientes da equação polinomial.

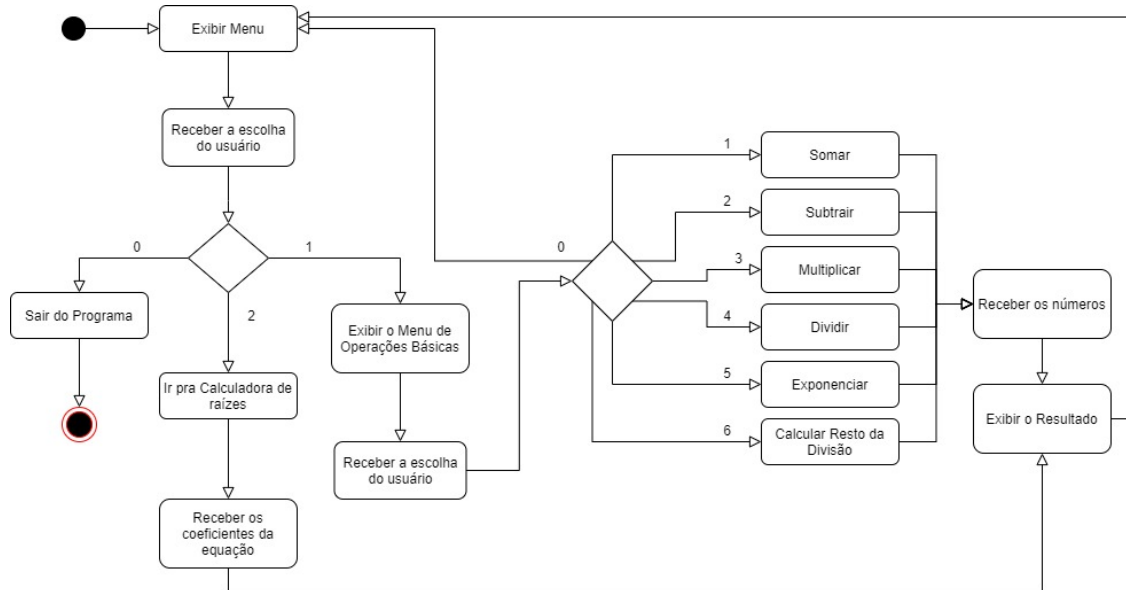
### **2.3. Diagrama de Atividades**

Um diagrama de atividade ilustra a dinâmica de um sistema pela modelagem do fluxos de controle e pois isso tipicamente, esses diagramas são usados para modelar fluxos de processos, processos de negócios ou operações internas. o diagrama de é similar ao funcionamento de uma máquina de estados (mas com o objetivo diferente), o qual envolve captura de ações e seus resultados em termos das mudanças no estado do objeto.

Geralmente representado por um gráfico de atividades, nele é mostrado o fluxo de uma atividade para outra (Uma atividade representa uma operação em alguma classe no sistema que resulta em uma mudança no estado do sistema). Esse fluxo é mostrado através de transições, que são setas direcionadas, de modo a mostrar o caminho entre os estados de atividade (ação).

E, os elementos do diagrama de atividades podem ser:

- Atividades (ações);
- Estados de atividade (ação);
- Transição;
- Fluxo de objeto;
- Estado inicial;
- Estado final;
- Branching;
- sincronização;
- Raias.



**Figura 3. Sistema CalculadoraC, diagrama de Atividades**

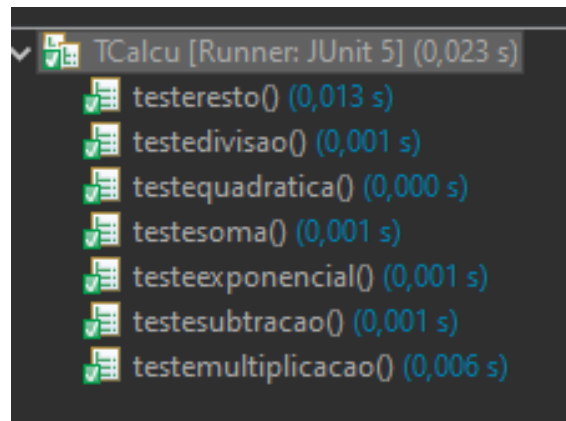
O diagrama acima representa, com um certo nível de abstração, todas as principais ações feitas pelo Sistema.

E, como pode ser observado no diagrama, o programa fica em loop até que o Usuário escolha sair.

### 3. Testes Computacionais

Para garantir a confiabilidade da calculadora e no resultado final, foram feitos testes automatizados com cada método com saídas esperadas. Pois quando implementados corretamente, estarão de acordo com a idealização prevista no diagrama inicial. Portanto utilizando o JUnit 5 foi possível a execução dos métodos buscando sempre a qualidade da sua execução.

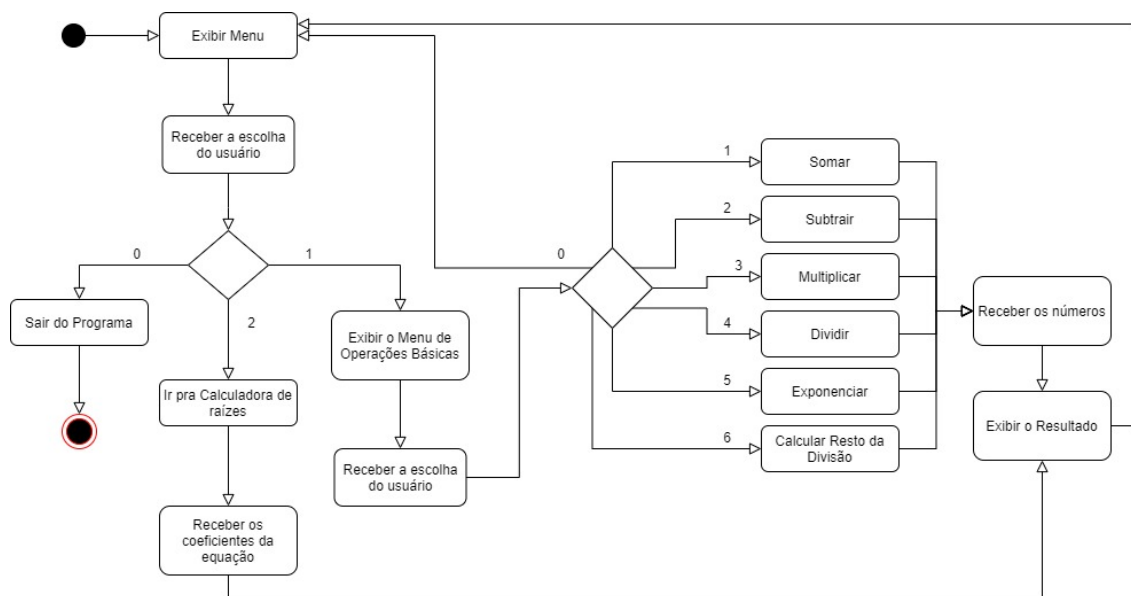
Cada método é testado e comparado com o resultado esperado. Para que isso seja possível, são adicionado os valores nos métodos de teste, o resultado esperado e assim realizar as operações de calculo. O JUnit mostra no seu console os testes conclusos. Caso haja alguma discrepância no esperado com o resultado real, os ajustes serão feitos no código fonte sem a necessidade de executar driver principal para verificar a qualidade do código. Economizando tempo em casos de testes com bastantes métodos.



```

TCalcu [Runner: JUnit 5] (0,023 s)
✓ testresto() (0,013 s)
✓ testedivisao() (0,001 s)
✓ testequadratica() (0,000 s)
✓ testesoma() (0,001 s)
✓ testeexponencial() (0,001 s)
✓ testesubtracao() (0,001 s)
✓ testemultiplicacao() (0,006 s)
  
```

**Figura 4. Relatório dos testes realizados pelo JUnit**



**Figura 5. Primeira parte, código de implementação de testes+**

Tendo em vista o diagrama apresentado, a seguir será compartilhado o código utilizado nos teste feitos:

```

import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertEquals;
class TCalcu {

    @Test
    public void testesoma() {
        CalculadoraC calc = new CalculadoraC(2, 3);
        double resultado = calc.soma();
        double esperada = 5;
        assertEquals(resultado, esperada, 0);
        if (resultado==esperada) {
            System.out.println("Soma funcionando");
        }
        else {
            System.out.println("Soma não está funcionando");
        }
    }

    @Test
    public void testesubtracao() {
        CalculadoraC calc = new CalculadoraC(80, 75);
        double resultado = calc.subt();
        double esperada = 5;
        assertEquals(resultado, esperada, 0);
        if (resultado==esperada) {
            System.out.println("Subtração funcionando");
        }
        else {
            System.out.println("Subtração não está funcionando");
        }
    }
}

```

Figura 6. Classe TCalcu, métodos testesoma e testesubtracao

```

import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertEquals;
class TCalcu {

    @Test
    public void testesoma() {
        CalculadoraC calc = new CalculadoraC(2, 3);
        double resultado = calc.soma();
        double esperada = 5;
        assertEquals(resultado, esperada, 0);
        if (resultado==esperada) {
            System.out.println("Soma funcionando");
        }
        else {
            System.out.println("Soma não está funcionando");
        }
    }

    @Test
    public void testesubtracao() {
        CalculadoraC calc = new CalculadoraC(80, 75);
        double resultado = calc.subt();
        double esperada = 5;
        assertEquals(resultado, esperada, 0);
        if (resultado==esperada) {
            System.out.println("Subtração funcionando");
        }
        else {
            System.out.println("Subtração não está funcionando");
        }
    }
}

```

Figura 7. Classe TCalcu, métodos testemultiplicacao e testedivisap



```

import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertEquals;
class TCalcu {

    @Test
    public void testesoma() {
        CalculadoraC calc = new CalculadoraC(2, 3);
        double resultado = calc.soma();
        double esperada = 5;
        assertEquals(resultado, esperada, 0);
        if (resultado==esperada) {
            System.out.println("Soma funcionando");
        }
        else {
            System.out.println("Soma não está funcionando");
        }
    }

    @Test
    public void testesubtracao() {
        CalculadoraC calc = new CalculadoraC(80, 75);
        double resultado = calc.subt();
        double esperada = 5;
        assertEquals(resultado, esperada, 0);
        if (resultado==esperada) {
            System.out.println("Subtração funcionando");
        }
        else {
            System.out.println("Subtração não está funcionando");
        }
    }
}

```

**Figura 8. Classe TCalcu, métodos teste exponencial e teste de restos**

## 4. Conclusão

A POO (Programação Orientada a Objetos) é um paradigma de linguagem que surgiu como uma alternativa a programação estruturada, esse paradigma baseia-se basicamente em dois conceitos chave: classes e objetos, sendo seus outros conceitos também importantes mas fundamentados neles. Além disso, Java é uma linguagem de programação orientada a objetos nascida na década de 90 e diferente das linguagens de programação modernas que são compiladas para código nativo, ela é compilada para um bytecode que é interpretado por uma máquina virtual (Java Virtual Machine, JVM).

Fazendo uma síntese das tecnologias desenvolvidas torna-se possível a criação da calculadora básica e de 2º grau, de forma a agregar a exploração de conceitos tais, como o uso devido de classes e instanciações, além de estabelecer a lógica de programação capaz de tornar o objetivo da calculadora (de resolver os problemas matemáticos) possíveis.

Contudo, tendo em vista este projeto é possível melhoramentos futuros, como adição de interface gráfica, de forma a tornar a calculadora um sistema cada vez mais completo. Também pode-se pensar em acrescentar análise de gráfico, histórico, dentre outras funcionalidades.

### Apêndice A – Código Fonte

Conclui-se, com base no código fonte (apêndice A), que a calculadora proposta tem as Classes Main e CalculadoraC.

```

import java.util.Scanner;

public class Main {
    private static Scanner input = new Scanner(System.in); // scanner global
    public static void main(String args[]) {
        boolean exec = true; // mantem o programa executando

        while(exec) {
            exibirMenu(1); // exibir menus com função para facilitar a legibilidade
            int escolha = input.nextInt();
            if (escolha == 0) {
                exec = false;
            } else {
                switch (escolha) {
                    case 1:
                        calcJava();
                        break;

                    case 2:
                        calcRaizes();
                        break;

                    default:
                        System.out.println("Opção inválida, escolha novamente\n");
                }
            }
        }

        input.close();
    }
}

```

**Figura 9. Classe Main, parte 1**

```

public static void calcJava() {
    int op = 0;
    double n1, n2;

    exibirMenu(2);
    op = input.nextInt(); // ler operação

    if (op == 0) { // caso o usuario deseje sair
        return;
    }

    System.out.println("Insira os numeros (um por linha):");
    n1 = input.nextDouble();
    n2 = input.nextDouble();

    CalculadoraC calc = new CalculadoraC(n1, n2);
    System.out.print("Resultado: ");

    switch (op) { // executa a função correspondente à operação escolhida pelo usuário
        case 1:
            System.out.printf("%f\n", calc.soma());
            break;

        case 2:
            System.out.printf("%f\n", calc.subt());
            break;

        case 3:
            System.out.printf("%f\n", calc.mult());
            break;

        case 4:
            System.out.printf("%f\n", calc.div());
            break;
    }
}

```

**Figura 10. Classe Main, parte 2**

```

        case 5:
            System.out.printf("%f\n", calc.exp());
            break;

        case 6:
            System.out.printf("%d\n", calc.rest());
            break;

        default:
            System.out.println("Opção inválida, escolha novamente\n");
    }

    System.out.println(); // newline extra para separar o próximo menu do resultado atual
}

public static void calcRaizes() {
    double a, b, c;
    System.out.println("Insira o valor de a: ");
    a = input.nextDouble();
    System.out.println("Insira o valor de b: ");
    b = input.nextDouble();
    System.out.println("Insira o valor de c: ");
    c = input.nextDouble();

    CalculadoraC calc = new CalculadoraC(a, b, c);

    double[] raizes = calc.root_quadraticEq();

    System.out.printf("Raizes:\nx1: %f\nx2: %f\n\n", raizes[0], raizes[1]);
}

```

**Figura 11. Classe Main, parte 3**

```

public static void exibirMenu(int idMenu) {
    switch (idMenu){
        case 1:
            System.out.println("-----Calculadora Científica-----");
            System.out.println("Escolha uma opção: ");
            System.out.println("1 - ir para a calculadora java");
            System.out.println("2 - ir para a calculadora de raizes");
            System.out.println("0 - Sair");
            break;
        case 2:
            System.out.println("---Calculadora Java---");
            System.out.println("Escolha uma operação:");
            System.out.println("1 - Adição");
            System.out.println("2 - Subtração");
            System.out.println("3 - Multiplicação");
            System.out.println("4 - Divisão");
            System.out.println("5 - Exponenciação");
            System.out.println("6 - Resto da divisão");
            System.out.println("0 - Sair");
            break;
    }
}
}
}

```

**Figura 12. Classe Main, parte 4**

```

public class _CalculadoraC {
    private double n1, n2;
    private double a, b, c;

    public CalculadoraC(double n1, double n2) { // construtor para operações normais
        this.n1 = n1;
        this.n2 = n2;
    }

    public CalculadoraC(double a, double b, double c) { // construtor para equações quadráticas
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public double soma() { return n1 + n2; }

    public double subtr() { return n1 - n2; }

    public double mult() { return n1 * n2; }

    public double div() { // retorna a divisão de n1 e n2
        try {
            return n1 / n2;
        } catch (RuntimeException e) { // levanta exceção caso o usuário tente dividir por zero
            throw new RuntimeException("Impossível dividir por 0.");
        }
    }

    public double exp() { // retorna exponenciação de n1 e n2
        if (n1 == 0 && n2 == 0) { // levanta exceção caso ambos números sejam zero
            throw new RuntimeException("Valor indefinido.");
        }
        return (double) Math.pow(n1, n2);
    }
}

```

Figura 13. Classe CalculadoraC, parte 1

```

    public int rest() { // retorna o resto da divisão entre n1 e n2
        try {
            return (int) (n1 % n2);
        } catch (RuntimeException e) { // levanta exceção caso o usuário tente dividir por zero
            throw new RuntimeException("Impossível dividir por 0.");
        }
    }

    public double[] root_quadraticEq() { // retorna as raízes da equação quadrática  $ax^2+bx+c$ 
        try {
            double[] x = new double[2]; // array de retorno
            x[0] = (double) ((-b + Math.sqrt(b * b - 4 * a * c)) / 2 * a); // calcula as raízes e faz cast para double
            x[1] = (double) ((-b - Math.sqrt(b * b - 4 * a * c)) / 2 * a);
            return x;
        } catch (RuntimeException e) { // levanta exceção caso a seja zero ou delta ( $b^2 - 4ac$ ) seja menor que zero
            throw new RuntimeException("Não há raízes reais para a equação quadrática! ");
        }
    }

    public void setN1(double n1) { this.n1 = n1; }

    public void setN2(double n2) { this.n2 = n2; }
}

```

Figura 14. Classe CalculadoraC, parte 2