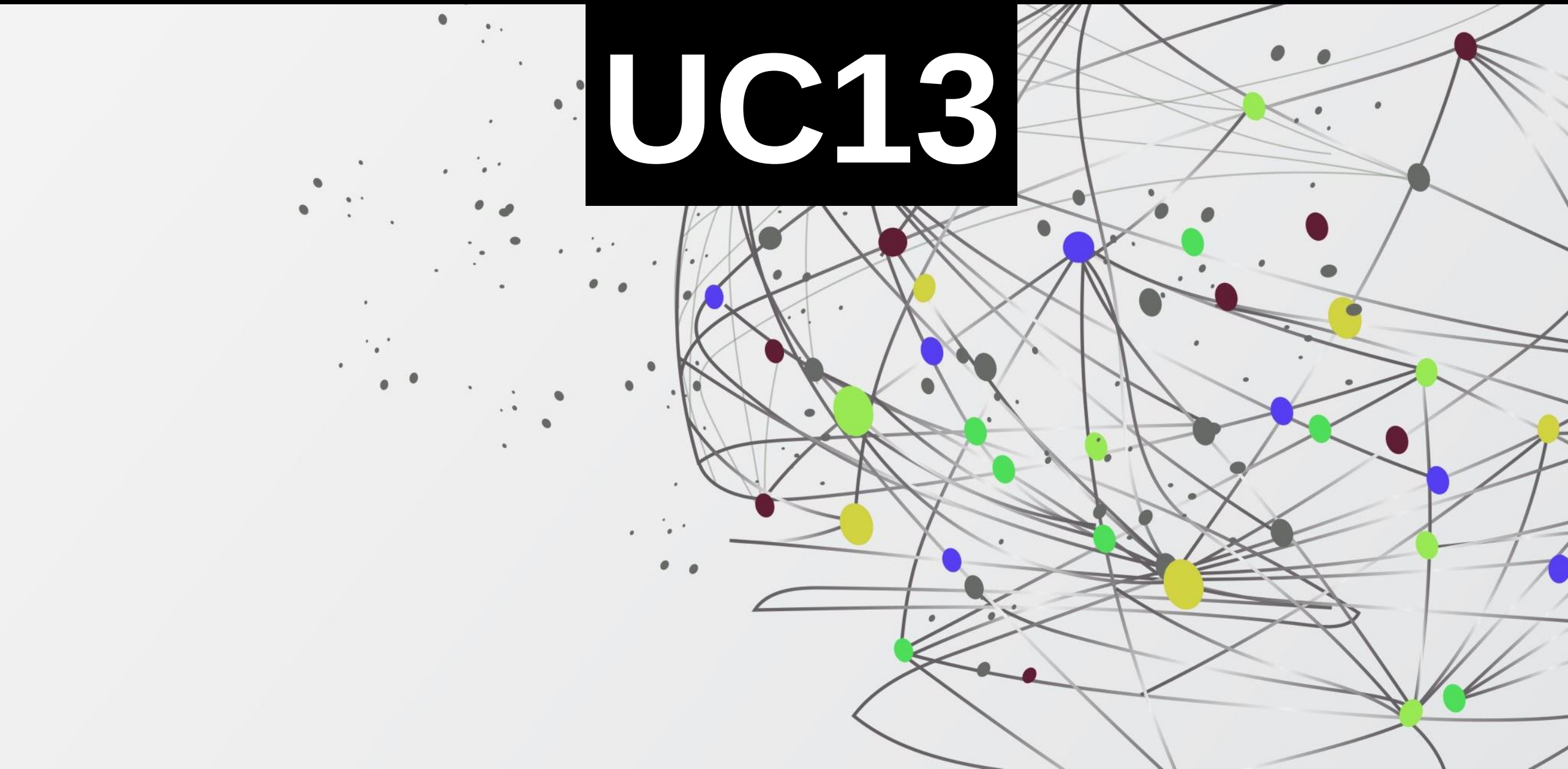
Executar os processos de codificação, manutenção e documentação de aplicativos computacionais para internet

UC13



Indicadores

- 1. Configura o ambiente de desenvolvimento conforme as funcionalidades e características do aplicativo computacional para WEB.

- 2. Desenvolve softwares de acordo com as melhores práticas da linguagem de programação selecionada.

- 3. Elabora código conforme as funcionalidades e características do aplicativo computacional para WEB.

- 4. Realiza a compilação e depuração do código de acordo com orientações técnicas da IDE utilizada.

- 5. Utiliza comandos de integração dos objetos de bancos de dados com o código construído para WEB de acordo com premissas do sistema operacional (servidor) de rede.

- 6. Elabora o manual do projeto de software desenvolvido conforme orientação técnica.

Conhecimento

Ferramentas de desenvolvimento de programas para internet. Ferramentas de desenvolvimento colaborativo. Ferramentas de modelagem de software.

Linguagens de programação. Ambientes de programação (IDE). · Linguagem de programação orientada a objetos – Visão geral da linguagem de programação. Palavras reservadas. Application Program Interface (API). Plataforma de desenvolvimento: internet. Tipos de dados. Variáveis e constantes. Coleções: lista, conjunto e mapa. Operadores. Comandos condicionais. Comandos de repetição. Objetos, classes, interfaces, atributos, modificadores de acesso, métodos e propriedades. Herança, polimorfismo, encapsulamento e agregação. Tratamento de erros e exceções. Distribuição do aplicativo. Defeitos e falhas. Documentação de programas de computador.

Controle de versão de software web – Segurança da informação. Instalação e configuração

Ferramentas de desenvolvimento de programas para internet. Ferramentas de desenvolvimento colaborativo. Ferramentas de modelagem de software.

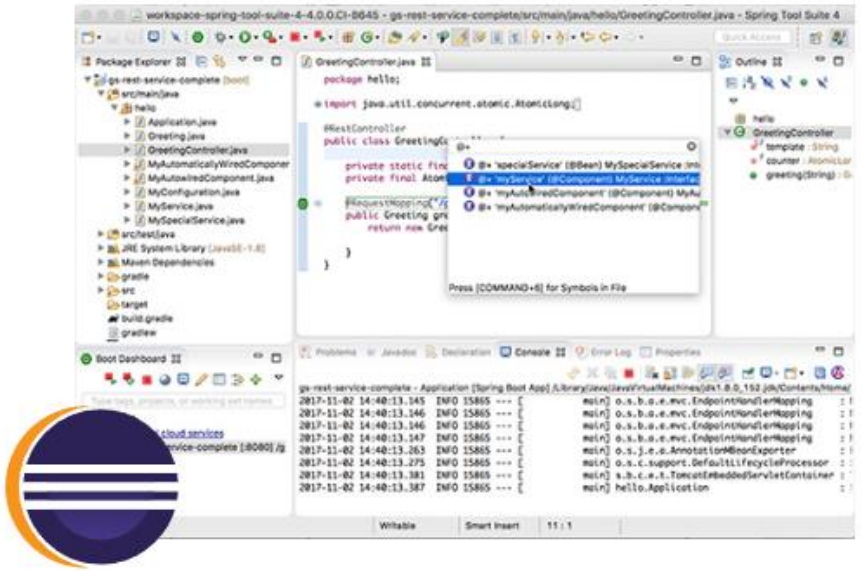
Ambientes de programação (IDE).

https://download.springsource.com/release/STS4/4.18.1.RELEASE/dist/e4.27/spring-tool-suite-4-4.18.1.RELEASE-e4.27.0-win32.win32.x86_64.self-extracting.jar

Spring Tools 4 for Eclipse

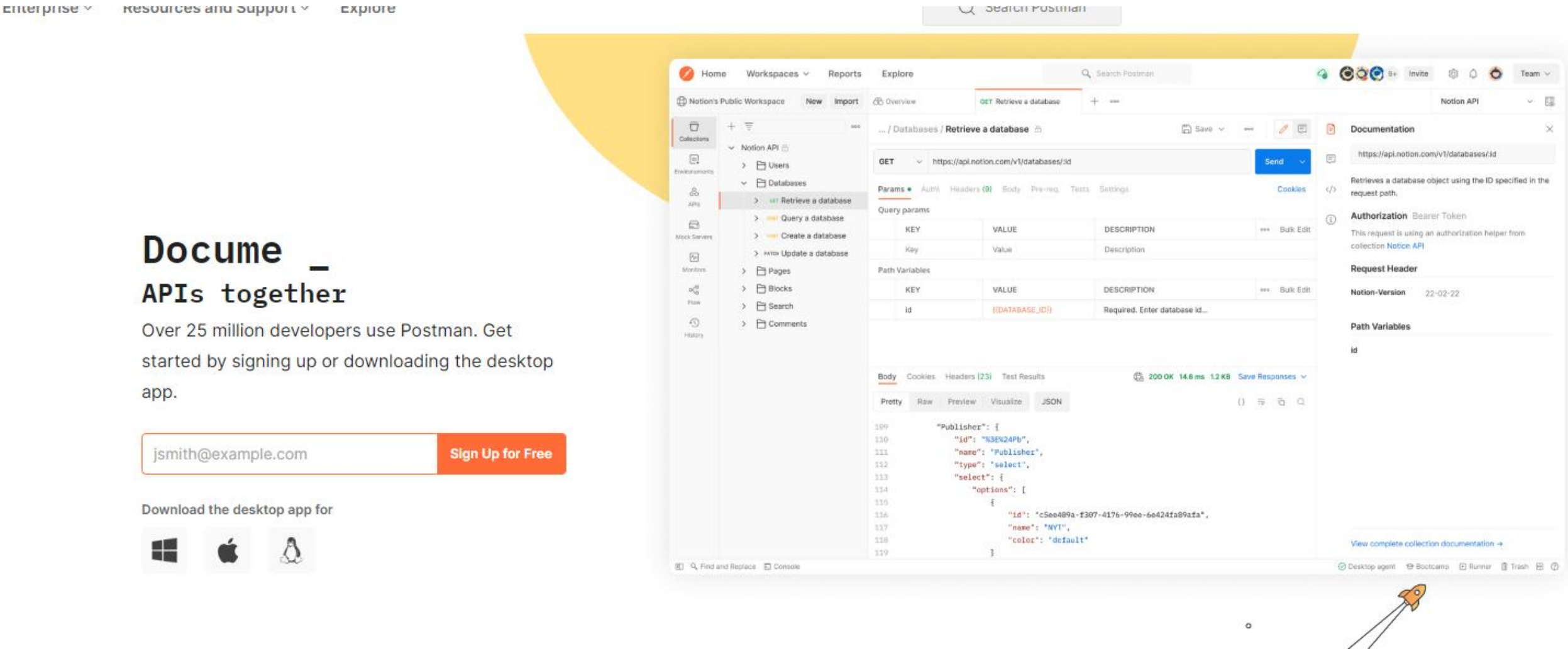
The all-new Spring Tool Suite 4. Free. Open source.

- 4.18.1 - LINUX X86_64
- 4.18.1 - LINUX ARM_64
- 4.18.1 - MACOS X86_64
- 4.18.1 - MACOS ARM_64
- 4.18.1 - WINDOWS X86_64



Instalação do Postman

https://www.postman.com/downloads/?utm_source=postman-home



Java

Crie um novo projeto chamado Pratica e neste crie todos os exemplos que vem a seguir. Os arquivos não tem o método main devem ser colocados em um pacote chamado classes e os que tem o main devem ser colocados em objetos. Rode apenas os objetos

```
3  public class Student
4  {
5     private String name;
6     private double average;
7
8     // construtor inicializa variáveis de instância
9     public Student(String name, double average)
10    {
11       this.name = name;
12
13       // valida que a média é > 0.0 e <= 100.0; caso contrário,
14       // armazena o valor padrão da média da variável de instância (0.0)
15       if (average > 0.0)
16          if (average <= 100.0)
17             this.average = average; // atribui à variável de instância
18    }
19
20    // define o nome de Student
21    public void setName(String name)
22    {
23       this.name = name;
24    }
25
26    // recupera o nome de Student
27    public String getName()
28    {
29       return name;
30    }
31
32    // define a média de Student
33    public void setAverage(double studentAverage)
34    {
35       // valida que a média é > 0.0 e <= 100.0; caso contrário,
36       // armazena o valor atual da média da variável de instância
37       if (average > 0.0)
38          if (average <= 100.0)
39             this.average = average; // atribui à variável de instância
40    }
41
42    // recupera a média de Student
43    public double getAverage()
44    {
45       return average;
46    }
47
48    // determina e retorna a letra da nota de Student
49    public String getLetterGrade()
50    {
51       String letterGrade = ""; // inicializado como uma String vazia
52
53       if (average >= 90.0)
54          letterGrade = "A";
55       else if (average >= 80.0)
56          letterGrade = "B";
57       else if (average >= 70.0)
58          letterGrade = "C";
59       else if (average >= 60.0)
60          letterGrade = "D";
61       else
62          letterGrade = "F";
63
64       return letterGrade;
65    }
66 } // finaliza a classe Student
```

Classe StudentTest

Para demonstrar as instruções `if...else` aninhadas no método `getLetterGrade` da classe `Student`, o método `main` da classe `StudentTest` (Figura 4.5) cria dois objetos `Student` (linhas 7 e 8). Então, as linhas 10 a 13 exibem o nome e a letra da nota de cada `Student` chamando os métodos `getName` e `getLetterGrade` dos objetos, respectivamente.

```
1  // Figura 4.5: StudentTest.java
2  // Cria e testa objetos Student.
3  public class StudentTest
4  {
5     public static void main(String[] args)
6     {
7        Student account1 = new Student("Jane Green", 93.5);
8        Student account2 = new Student("John Blue", 72.75);
9
10       System.out.printf("%s's letter grade is: %s%n",
11          account1.getName(),account1.getLetterGrade());
12       System.out.printf("%s's letter grade is: %s%n",
13          account2.getName(),account2.getLetterGrade());
14    }
15 } // fim da classe StudentTest
```

Implementando repetição controlada por contador

Na Figura 4.8, o método `main` da classe `ClassAverage` (linhas 7 a 31) implementa o algoritmo para calcular a média da classe descrita pelo pseudocódigo na Figura 4.7 — ele permite que o usuário insira 10 notas, então, calcula e exibe a média.

```
1  // Figura 4.8: ClassAverage.java
2  // Resolvendo o problema da média da classe usando a repetição controlada por contador.
3  import java.util.Scanner; // programa utiliza a classe Scanner
4
```

continua

```
5  public class ClassAverage
6  {
7     public static void main(String[] args)
8     {
9        // cria Scanner para obter entrada a partir da janela de comando
10       Scanner input = new Scanner(System.in);
11
12       // fase de inicialização
13       int total = 0;  // inicializa a soma das notas inseridas pelo usuário
14       int gradeCounter = 1; // inicializa nº da nota a ser inserido em seguida
15
16       // fase de processamento utiliza repetição controlada por contador
17       while (gradeCounter <= 10) // faz o loop 10 vezes
18       {
19          System.out.print("Enter grade: "); // prompt
20          int grade = input.nextInt(); // insere a próxima nota
21          total = total + grade; // adiciona grade a total
22          gradeCounter = gradeCounter + 1; // incrementa o contador por 1
23       }
24
25       // fase de término
26       int average = total / 10; // divisão de inteiros produz um resultado inteiro
27
28       // exibe o total e a média das notas
29       System.out.printf("%nTotal of all 10 grades is %d%n", total);
30       System.out.printf("Class average is %d%n", average);
31    }
32 } // fim da classe ClassAverage
```

```java
// Resolvendo o problema da média da classe usando a repetição controlada por sentinela.
import java.util.Scanner; // programa utiliza a classe Scanner

public class ClassAverage
{
   public static void main(String[] args)
   {
      // cria Scanner para obter entrada a partir da janela de comando
      Scanner input = new Scanner(System.in);

      // fase de inicialização
      int total = 0; // incializa a soma das notas
      int gradeCounter = 0; // inicializa o nº de notas inseridas até agora

      // fase de processamento
      // solicita entrada e lê a nota do usuário
      System.out.print("Enter grade or -1 to quit: ");
      int grade = input.nextInt();

      // faz um loop até ler o valor de sentinela inserido pelo usuário
      while (grade != -1)
      {
         total = total + grade; // adiciona grade a total
         gradeCounter = gradeCounter + 1; // incrementa counter

         // solicita entrada e lê a próxima nota fornecida pelo usuário
         System.out.print("Enter grade or -1 to quit: ");
         grade = input.nextInt();
      }

      // fase de término
      // se usuário inseriu pelo menos uma nota...
      if (gradeCounter != 0)
      {
         // usa número com ponto decimal para calcular média das notas
         double average = (double) total / gradeCounter;

         // exibe o total e a média (com dois dígitos de precisão)
         System.out.printf("%nTotal of the %d grades entered is %d%n",
            gradeCounter, total);
         System.out.printf("Class average is %.2f%n", average);
      }
      else // nenhuma nota foi inserida, assim gera a saída da mensagem apropriada
         System.out.println("No grades were entered");
   }
} // fim da classe ClassAverage
```

```java
// Análise dos resultados do exame utilizando instruções de controle aninhadas.
import java.util.Scanner; // classe utiliza a classe Scanner

public class Analysis
{
   public static void main(String[] args)
   {
      // cria Scanner para obter entrada a partir da janela de comando
      Scanner input = new Scanner(System.in);

      // inicializando variáveis nas declarações
      int passes = 0;
      int failures = 0;
      int studentCounter = 1;

      // processa 10 alunos utilizando o loop controlado por contador
      while (studentCounter <= 10)
      {
         // solicita ao usuário uma entrada e obtém valor fornecido pelo usuário
         System.out.print("Enter result (1 = pass, 2 = fail): ");
         int result = input.nextInt();

         // if...else está aninhado na instrução while
         if (result == 1)
            passes = passes + 1;
         else
            failures = failures + 1;

         // incrementa studentCounter até o loop terminar
         studentCounter = studentCounter + 1;
      }

      // fase de término; prepara e exibe os resultados
      System.out.printf("Passed: %d%nFailed: %d%n", passes, failures);

      // determina se mais de 8 alunos foram aprovados
      if (passes > 8)
         System.out.println("Bonus to instructor!");
   }
} // fim da classe Analysis
```

```
2  // Operadores de pré-incremento e de pós-incremento.
3
4  public class Increment
5  {
6     public static void main(String[] args)
7     {
8        // demonstra o operador de pós-incremento
9        int c = 5;
10       System.out.printf("c before postincrement: %d%n", c); // imprime 5
11       System.out.printf("    postincrementing c: %d%n", c++); // imprime 5
12       System.out.printf(" c after postincrement: %d%n", c); // imprime 6
13
14       System.out.println(); // pula uma linha
15
16       // demonstra o operador de pré-incremento
17       c = 5;
18       System.out.printf(" c before preincrement: %d%n", c); // imprime 5
19       System.out.printf("     preincrementing c: %d%n", ++c); // imprime 6
20       System.out.printf("  c after preincrement: %d%n", c); // imprime 6
21    }
22 } // fim da classe Increment
```

```java
// Utilizando DrawLine para conectar os cantos de um painel.
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawPanel extends JPanel
{
   // desenha um X a partir dos cantos do painel
   public void paintComponent(Graphics g)
   {
      // chama paintComponent para assegurar que o painel é exibido corretamente
      super.paintComponent(g);

      int width = getWidth(); // largura total
      int height = getHeight(); // altura total

      // desenha uma linha a partir do canto superior esquerdo até o inferior direito
      g.drawLine(0, 0, width, height);

      // desenha uma linha a partir do canto inferior esquerdo até o superior direito
      g.drawLine(0, height, width, 0);
   }
} // fim da classe DrawPanel
```

```
2  // Criando JFrame para exibir um DrawPanel.
3  import javax.swing.JFrame;
4
5  public class DrawPanelTest
6  {
7     public static void main(String[] args)
8     {
9        // cria um painel que contém nosso desenho
10       DrawPanel panel = new DrawPanel();
11
12       // cria um novo quadro para armazenar o painel
13       JFrame application = new JFrame();
14
15       // configura o frame para ser encerrado quando ele é fechado
16       application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17
18       application.add(panel); // adiciona o painel ao frame
19       application.setSize(250, 250); // configura o tamanho do frame
20       application.setVisible(true); // torna o frame visível
21    }
22 } // fim da classe DrawPanelTest
```

Instrução de repetição for

A Seção 5.2 apresentou os princípios básicos da repetição controlada por contador. A instrução `while` pode ser utilizada para implementar qualquer loop controlado por contador. O Java também fornece a **instrução de repetição `for`**, que especifica os detalhes da repetição controlada por contador em uma única linha de código. A Figura 5.2 reimplementa o aplicativo da Figura 5.1 utilizando `for`.

```
1  // Figura 5.2: ForCounter.java
2  // Repetição controlada por contador com a instrução de repetição for.
3
4  public class ForCounter
5  {
6     public static void main(String[] args)
7     {
8        // o cabeçalho da instrução for inclui inicialização,
9        // condição de continuação do loop e incremento
10       for (int counter = 1; counter <= 10; counter++)
11          System.out.printf("%d  ", counter);
12
13       System.out.println();
14    }
15 } // fim da classe ForCounter
```

Aplicativo: somando os inteiros pares de 2 a 20

Agora consideramos dois aplicativos de exemplo que demonstram as utilizações simples de `for`. O aplicativo na Figura 5.5 utiliza uma instrução `for` para somar os inteiros pares de 2 a 20 e armazenar o resultado em uma variável `int` chamada `total`.

```
1  // Figura 5.5: Sum.java
2  // Somando inteiros com a instrução for.
3
4  public class Sum
5  {
6     public static void main(String[] args)
7     {
8        int total = 0;
9
10       // total de inteiros pares de 2 a 20
11       for (int number = 2; number <= 20; number += 2)
12          total += number;
13
14       System.out.printf("Sum is %d%n", total);
15    }
16 } // fim da classe Sum
```

```java
// Cálculos de juros compostos com for.

public class Interest
{
   public static void main(String[] args)
   {
      double amount; // quantia em depósito ao fim de cada ano
      double principal = 1000.0; // quantidade inicial antes dos juros
      double rate = 0.05; // taxa de juros

      // exibe cabeçalhos
      System.out.printf("%s%20s %n", "Year", "Amount on deposit");

      // calcula quantidade de depósito para cada um dos dez anos
      for (int year = 1; year <= 10; ++year)
      {
         // calcula nova quantidade durante ano especificado
         amount = principal * Math.pow(1.0 + rate, year);

         // exibe o ano e a quantidade
         System.out.printf("%4d%,20.2f%n", year, amount);
      }
   }
} // fim da classe Interest
```

Quando a instrução `if` aninhada nas linhas 11 e 12 da instrução `for` (linhas 9 a 15) detecta que `count` é 5, a instrução `break` na linha 12 é executada. Isso termina a instrução `for` e o programa prossegue para a linha 17 (imediatamente depois da instrução `for`), que exibe uma mensagem que indica o valor da variável de controle quando o loop terminar. O loop executa completamente o seu corpo somente quatro vezes em vez de 10.

```
1  // Figura 5.13: BreakTest.java
2  // a instrução break sai de uma instrução for.
3  public class BreakTest
4  {
5     public static void main(String[] args)
6     {
7        int count; // variável de controle também utilizada depois que loop termina
8
9        for (count = 1; count <= 10; count++) // faz o loop 10 vezes
10       {
11          if (count == 5)
12             break; // termina o loop se a contagem for 5
13
14          System.out.printf("%d ", count);
15       }
16
17       System.out.printf("%nBroke out of loop at count = %d%n", count);
18    }
19 } // fim da classe BreakTest
```

Instrução `continue`

A instrução `continue`, quando executada em um `while`, `for` ou `do...while`, pula as instruções restantes no corpo do loop e prossegue com a *próxima iteração* do loop. Nas instruções `while` e `do...while`, o programa avalia o teste de continuação do loop imediatamente depois que a instrução `continue` é executada. Em uma instrução `for`, a expressão incremento é executada, então o programa avalia o teste de continuação do loop.

A Figura 5.14 utiliza `continue` (linha 10) para pular para a instrução na linha 12 quando o `if` aninhado determina que o valor de `count` é `5`. Quando a instrução `continue` executa, o controle de programa continua com o incremento da variável de controle na instrução `for` (linha 7).

Na Seção 5.3, declaramos que `while` poderia ser utilizado na maioria dos casos no lugar de `for`. Isso *não* é verdade quando a expressão de incremento no `while` segue-se a uma instrução `continue`. Nesse caso, o incremento *não* executa antes de o programa avaliar a condição de continuação da repetição, então o `while` não é executado da mesma maneira que o `for`.

```java
// Figura 5.14: ContinueTest.java
// Instrução continue que termina uma iteração de uma instrução for.
public class ContinueTest
{
   public static void main(String[] args)
   {
      for (int count = 1; count <= 10; count++) // faz o loop 10 vezes
      {
         if (count == 5)
            continue; // pula o código restante no corpo do loop se a contagem for 5

         System.out.printf("%d ", count);
      }

      System.out.printf("%nUsed continue to skip printing 5%n");
   }
} // fim da classe ContinueTest
```

```java
// Figura 5.14: ContinueTest.java
// Instrução continue que termina uma iteração de uma instrução for.
public class ContinueTest
{
   public static void main(String[] args)
   {
      for (int count = 1; count <= 10; count++) // faz o loop 10 vezes
      {
         if (count == 5)
            continue; // pula o código restante no corpo do loop se a contagem for 5

         System.out.printf("%d ", count);
      }

      System.out.printf("%nUsed continue to skip printing 5%n");
   }
} // fim da classe ContinueTest
```

```java
// Figura 5.27: Shapes.java
// Desenhando uma cascata de formas com base na escolha do usuário.
import java.awt.Graphics;
import javax.swing.JPanel;

public class Shapes extends JPanel
{
   private int choice; // escolha do usuário de qual forma desenhar

   // construtor configura a escolha do usuário
   public Shapes(int userChoice)
   {
      choice = userChoice;
   }

   // desenha uma cascata de formas que iniciam do canto superior esquerdo
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      for (int i = 0; i < 10; i++)
      {
         // seleciona a forma com base na escolha do usuário
         switch (choice)
         {
            case 1: // desenha retângulos
               g.drawRect(10 + i * 10, 10 + i * 10,
                  50 + i * 10, 50 + i * 10);
               break;
            case 2: // desenha ovais
               g.drawOval(10 + i * 10, 10 + i * 10,
                  50 + i * 10, 50 + i * 10);
               break;
         }
      }
   }
} // fim da classe Shapes
```

```java
// Figura 5.28: ShapesTest.java
// Obtendo a entrada de usuário e criando um JFrame para exibir Shapes.
import javax.swing.JFrame; // manipula a exibição
import javax.swing.JOptionPane;

public class ShapesTest
{
   public static void main(String[] args)
   {
      // obtém a escolha do usuário
      String input = JOptionPane.showInputDialog(
         "Enter 1 to draw rectangles\n " +
         "Enter 2 to draw ovals");

      int choice = Integer.parseInt(input); // converte a entrada em int

      // cria o painel com a entrada do usuário
      Shapes panel = new Shapes(choice);

      JFrame application = new JFrame(); // cria um novo JFrame

      application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      application.add(panel);
      application.setSize(300, 300);
      application.setVisible(true);
   }
} // fim da classe ShapesTest
```

Declarando métodos com múltiplos parâmetros

Os métodos costumam exigir mais de uma informação para realizar suas tarefas. Agora, iremos considerar como escrever seus próprios métodos com *múltiplos* parâmetros.

A Figura 6.3 utiliza um método chamado `maximum` para determinar e retornar o maior dos três valores `double`. No `main`, as linhas 14 a 18 solicitam que o usuário insira três valores `double`, então os lê a partir do usuário. A linha 21 chama o método `maximum` (declarado nas linhas 28 a 41) para determinar o maior dos três valores que recebe como argumentos. Quando o método `maximum` retorna o resultado para a linha 21, o programa atribui o valor de retorno de `maximum` à variável local `result`. Em seguida, a linha 24 gera a saída do valor máximo. No final desta seção, discutiremos o uso do operador + na linha 24.

```
1  // Figura 6.3: MaximumFinder.java
2  // Método maximum declarado pelo programador com três parâmetros double.
3  import java.util.Scanner;
4
```

continua

```
5  public class MaximumFinder
6  {
7     // obtém três valores de ponto flutuante e localiza o valor máximo
8     public static void main(String[] args)
9     {
10       // cria Scanner para entrada a partir da janela de comando
11       Scanner input = new Scanner(System.in);
12
13       // solicita e insere três valores de ponto flutuante
14       System.out.print(
15          "Enter three floating-point values separated by spaces: ");
16       double number1 = input.nextDouble(); // lê o primeiro double
17       double number2 = input.nextDouble(); // lê o segundo double
18       double number3 = input.nextDouble(); // lê o terceiro double
19
20       // determina o valor máximo
21       double result = maximum(number1, number2, number3);
22
23       // exibe o valor máximo
24       System.out.println("Maximum is: " + result);
25    }
26
27    // retorna o máximo dos seus três parâmetros de double
28    public static double maximum(double x, double y, double z)
29    {
30       double maximumValue = x; // supõe que x é o maior valor inicial
31
32       // determina se y é maior que maximumValue
33       if (y > maximumValue)
34          maximumValue = y;
35
36       // determina se z é maior que maximumValue
37       if (z > maximumValue)
38          maximumValue = z;
39
40       return maximumValue;
41    }
42 } // fim da classe MaximumFinder
```

```java
// Figura 6.11: DrawSmiley.java
// Desenhando um rosto sorridente com cores e formas preenchidas.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawSmiley extends JPanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // desenha o rosto
      g.setColor(Color.YELLOW);
      g.fillOval(10, 10, 200, 200);

      // desenha os olhos
      g.setColor(Color.BLACK);
      g.fillOval(55, 65, 30, 30);
      g.fillOval(135, 65, 30, 30);

      // desenha a boca
      g.fillOval(50, 110, 120, 60);

      // "retoca" a boca para criar um sorriso
      g.setColor(Color.YELLOW);
      g.fillRect(50, 110, 120, 30);
      g.fillOval(50, 120, 120, 40);
   }
} // fim da classe DrawSmiley
```

```
1  // Figura 6.12: DrawSmileyTest.java
2  // Aplicativo de teste que exibe um rosto sorridente.
3  import javax.swing.JFrame;
4
5  public class DrawSmileyTest
6  {
7     public static void main(String[] args)
8     {
9        DrawSmiley panel = new DrawSmiley();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       application.add(panel);
14       application.setSize(230, 250);
15       application.setVisible(true);
16    }
17 } // fim da classe DrawSmileyTest
```

Continua ...