Atividade 2 - Áulus Pinho

Desafio 1

a) Não é possível executar a classe Circle por falta de um método main().
b) Após a criaçao e ediçao da classe TestCricle, a compilação foi feita com sucesso. Na declaração da *String color* porém, o Eclipse apontou um aviso de que esta variável não estava sendo usada. O motivo aparentou ser que não havia nenhum método diferente dos construtores que utilizam esta variável.
O resultado inicial, assim como o código encontram-se mostrados abaixo:

Resultado:

```
O circulo tem raio de 1.0 e area de 3.141592653589793
```

Códigos:

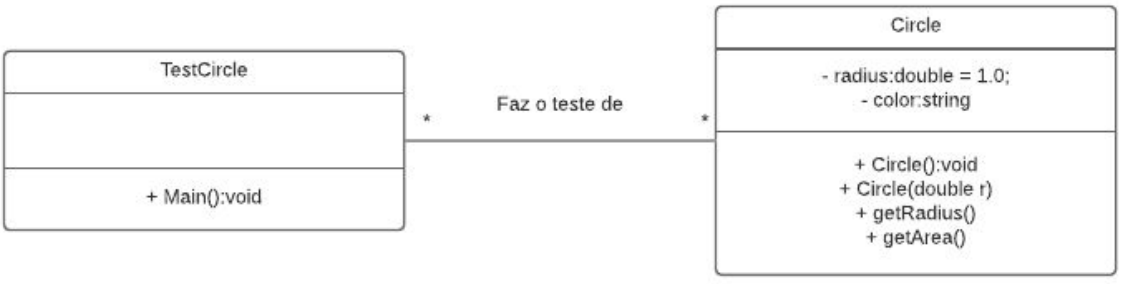
```
public class Circle { // salvar como "Circle.java"
 // variáveis de instancia privadas, isto é, não acessíveis de fora
desta classe.
 private double radius;
 private String color;
 // primeiro construtor o qual atribui valores iniciais a ambos: radius
e color .
 public Circle() {
 radius = 1.0;
 color = "red";
 }
 // Segundo construtor que inicia radius com o parâmetro recebido, e
matem color com
 // o valor padrão definido.
 public Circle(double r) {
 this( ); // cria o objeto com o primeiro construtor:.Circle()
 color = "red";
 }
 // Metodo de acesso para obter o valor armazenado em radius
 public double getRadius() {
 return radius;
 }
 // Metodo de acesso para computar a área de um circulo.
 public double getArea() {
 return radius*radius*Math.PI; // PI é a constante ¶. Math é a classe
onde PI é definido ¶.
```

```
 }
}

public class TestCircle { // salve como "TestCircle.java"
 public static void main(String[ ] args) {
 // Declara c1 como variável habilitada a armazenar uma referencia para
objeto da classe //Circle.
 Circle c1;
 // atribui a c1 .a referencia retornada pelo construtor padrão Circle
()
 c1 = new Circle();
 // Para invocar os metodos classe Circle para operar sobre a instância
c1,
 // usa-se o operador ponto (".").
 //Em outras palavras: usa-se o ponto para enviar uma mensagem ao objeto
c1 para que
 // ele execute um de seus métodos.
 System.out.println( "O circulo tem o raio de " + c1.getRadius()
 + " e area de " + c1.getArea() );

 // Declara e aloca uma segunda instancia da classe Circle chamada c2
 // com o valor do radius igual a 2.0 e color com valor padrão.
 Circle c2 = new Circle(2.0);
 // Para invocar os metodos a operar sobre a instância c2, usa-se o
operador ponto (".")
 System.out.println( "O circulo tem raio de " + c2.getRadius()
 + " e area de " + c2.getArea() );
 } // fim do método main()
} // fim da classe TestCircle
```

Diagrama de Classes



Desafio 2

1. Resultado:

```
O circulo tem o raio de 1.0 e area de 3.141592653589793
```

2. Resultado:

```
O circulo tem o raio de 1.0 e area de 3.141592653589793
O circulo tem raio de 1.0 e area de 3.141592653589793
O circulo tem raio de 2.0 e area de 12.566370614359172
```

3. Resultado:

```
O circulo tem o raio de 1.0 e area de 3.141592653589793
O circulo tem raio de 1.0 e area de 3.141592653589793
O circulo tem raio de 2.0 e area de 12.566370614359172
O circulo tem raio de 2.0, area de 12.566370614359172 e cor azul
```

4. Resultado:

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problem:
The field Circle.radius is not visible

at TestCircle.main(TestCircle.java:28)
```

Por ser uma variável privada, o acesso à ela por um método de outra classe não é possível. Por este motivo, a compilação do programa não é completada, apresentando a mensagem acima mostrada.

9.

a) Resultado:

```
O circulo tem o raio de 1.0 e area de 3.141592653589793
O circulo tem raio de 2.0 e area de 12.566370614359172
O circulo tem raio de 5.0 e area de 78.53981633974483
O circulo tem raio de 5.0, area de 78.53981633974483 e cor green
Circulo: raio = 1.0 cor = red
```

b) Resultado:

```
Circulo: raio = 1.2 cor = red
Circulo: raio = 1.2 cor = red
```

```
Aqui o operador '+' também invoca toString(): Circulo: raio = 1.2 cor =
red
```

O resultado é bem interessante. O método toString realmente foi executado, independente se a chamada era explícita ou implícita.
O código completo resultado de todas as mudanças feitas no Desafio 2 encontra-se exposto abaixo.

```
public class TestCircle {
// salve como "TestCircle.java"
      public static void main(String[] args) {
            // Declara c1 como variÃ¡vel habilitada a armazenar
uma referencia para objeto da classe Circle.
            Circle c1;

            // atribui a c1 .a referencia retornada pelo construtor
padrÃ£o Circle ()
            c1    = new Circle();

            /* Para invocar os metodos classe Circle para  operar sobre
a instÃ¢ncia c1,usa-se o operador ponto (â€œ.â€).
            Em outras palavras: usa-se o ponto para enviar uma mensagem
ao objeto c1 para que ele execute um de seus mÃ(c)todos.
            */

            System.out.println("O circulo tem o raio de "+
c1.getRadius() + " e area de "+ c1.getArea());
            /*
            Circle c2 = new Circle(1.2);

            System.out.println(c2.toString());  // chamada explicita

            System.out.println(c2);// println() chama toString()
implicitamente

            // no exemplo abaixo o operador '+' invoca c2.toString()
implicitamente.
            System.out.println("Aqui o operador '+' também invoca
toString(): " + c2);
            */
            System.out.println(c1.toString()); // chamada explicita

      }
      // fim do mÃ(c)todo  main()
}
```

Desafio 3

- a) Código da superclasse Point:

```
public class Point {
	private float x;
	private float y;
	public void setCoordenadas(int x, int y) {
		this.x = x;
		this.y = y;
	}
	public float getx() {
		return x;
	}
	public float gety() {
		return y;
	}
	public String toString() {
		return "Ponto: x = " + x + "y = " + y;
	}

}
```

b) Código da classe Circle

```
public class Circle extends Point {
	// variÃ¡veis de instancia privadas, isto Ã(c), nÃ£o acessÃveis de
fora desta classe.
	private double radius;
	private String color;
	private double length;
	//private Point center = new Point();
	// primeiro construtor o qual atribui valores iniciais a ambos:
radius e color .
	public Circle() {
		radius = 1.0;
		color = "red";
		length = 2*radius*Math.PI;
		super.setCoordenadas(0, 0);
	}

	// Segundo construtor que inicia radius com o parÃ¢metro recebido,
e matem color com o valor padrÃ£o definido.
```

```
	public Circle(double radius, int x, int y) {
		this.radius = radius;
		color = "red"; // cria o objeto com o primeiro
construtor:.Circle()
		super.setCoordenadas(x,y);
	}
	public Circle(double radius, String color, int x, int y){
		this.radius = radius;
		this.color = color;
		super.setCoordenadas(x,y);
	}
	public String toString() {
		return "Circulo: raio = " + radius + " cor = " + color + "
centro: x = " + super.getx() + " y = " + super.gety();
	}
	// metodo Set para a variÃ¡vel de instancia radius
	public void setRadius(double radius) {
		this.radius = radius;
	}
	// metdodo Set para a variÃ¡vel de instancia  color
	public void setColor(String color) {
		this.color = color;
	}
	public void setCircleCenter(int x, int y) {
		super.setCoordenadas(x,y);
	}
	// Metodo de acesso para obter o valor armazenado em radius
	public double getRadius() {
		return radius;
	}
	// Metodo de acesso para computar a Ã¡rea de um circulo.
	public double getArea() {
		return radius*radius*Math.PI; // PI Ã(c) a constante. Math
Ã(c) a classe onde PI Ã(c) definido
	}
	public double getLength() {
		return length;
	}
	public float getCircleCenter_x() {
		return super.getx();
	}
	public float getCircleCenter_y() {
		return super.gety();
	}
	public String getColor() {
```

```
            return color;
      }

}
```

- Código da Classe Triangle e TestTriangle

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Scanner;

public class Triangle {
      private double a;
      private double b;
      private double c;
      private String type;
      private Scanner resposta;
      private double angles[];
      public Triangle() {
            angles = new double[3];
            boolean triangle;
            do {
                  System.out.println("Digite 3 valores para os lados do
triangulo:");
                  resposta = new Scanner(System.in);
                  a = resposta.nextDouble();
                  b = resposta.nextDouble();
                  c = resposta.nextDouble();
                  triangle = MakeTriangle(a,b,c);
            }while(!triangle);
      }
      public boolean MakeTriangle(double a, double b, double c) {
            if(!IsValidTriangle(a, b, c))
                  if(!LadosErrados(a,b,c))
                        return false;
            else {
                  System.out.println("\\nOs lados especificados formam um
triangulo\n");
                  setTriangle(a,b,c);
                  System.out.println("\nTriangulo criado com sucesso\n");
            }
            return true;
      }
      private boolean LadosErrados(double a, double b, double c) {
            System.out.println("\nOs lados especificados nao formam um
triangulo\n" + "Escolha uma opcao: \n1 -> Definir outros valores\n2 -> Deixar
que o programa gere um triangulo");
            resposta = new Scanner(System.in);
            System.out.println("\nDigite o numero corrrespondente a opcao
```

```
escolhida:");
              int opcao = resposta.nextInt();
              if(opcao == 1)
                     return false;
              FazNovoTriangulo(a,b,c);
              System.out.println("\nO novo triangulo possui lados: "+"a =
"+this.a + "b = "+ this.b + "c = "+this.c + "\n");
              return true;
       }
       private void FazNovoTriangulo(double a, double b, double c) {
              do{
                     a++;
                     b++;
                     c++;
                     if(IsValidTriangle(a,b,c)) {
                            setTriangle(a,b,c);
                            return;
                     }
              }while(true);
       }
       public void setTriangle(double a, double b, double c) {
                     this.a = a;
                     this.b = b;
                     this.c = c;
       }
       private boolean IsValidTriangle(double a, double b, double c) {
              if(a >= b+c || b >= a+c || c >= a+b)
                     return false;
              return true;
       }
       public double Perimeter() {
              return a+b+c;
       }
       public double Area() {
              //segundo a formula de Herão de Alexandria:
              double per = Perimeter()/2;
              per *= (per-a)*(per-b)*(per-c);
              return Math.sqrt(per);
       }
       public void FindType(){
              Type_By_Sides();
              setAngles();
              Type_By_Angles();
       }
       private void Type_By_Sides(){
              if(a==b && b==c && c==a)
                     type = "Equilatero";
              else if(a==b || b==c || c==a)
                     type = "Isosceles";
              else if(a!=b && b!=c || c!=a)
```

```
                type = "Escaleno";
        }
        private void Type_By_Angles() {
                if(angles[0] < 90 || angles[1] < 90 || angles[2] < 90)
                        type += " e Acutangulo";
                else if(angles[0] == 90 || angles[1] == 90 || angles[2] == 90)
                        type += " e Retangulo";
                else if(angles[0] > 90 || angles[1] > 90 || angles[2] > 90)
                        type += " e Obtusangulo";
        }
        private void setAngles() {
                //a
                double a2 = Math.pow(a, 2);
                double b2 = Math.pow(b, 2);
                double c2 = Math.pow(c, 2);
                double cosAngle = (a2 - b2 - c2)/(-1*2*b*c);
                //angulo a
                angles[0] = Math.toDegrees(Math.acos(cosAngle));
                cosAngle = (b2 - a2 - c2)/(-1*2*a*c);
                //angulo b
                angles[1] = Math.toDegrees(Math.acos(cosAngle));
                //angulo c
                angles[2] = 180 - angles[0] - angles[1];
        }
        private void Truncate_Angles_Values() {
                angles[0] = BigDecimal.valueOf(angles[0]).setScale(4,
RoundingMode.HALF_UP).doubleValue();
                angles[1]= BigDecimal.valueOf(angles[1]).setScale(4,
RoundingMode.HALF_UP).doubleValue();
                angles[2] = BigDecimal.valueOf(angles[2]).setScale(4,
RoundingMode.HALF_UP).doubleValue();
        }
        public String toString() {
                Truncate_Angles_Values();
                return "Traingulo: Lados: a = " + a + " b = " + b + " c = " + c +
                                "\nArea: " + BigDecimal.valueOf(Area()).setScale(4,
RoundingMode.HALF_UP).doubleValue() + "\nPerimetro: " + Perimeter() + "\nTipo: "
+ type +
                                "\nAngulos Internos: " + angles[0] + " " + angles[1]
+ " " + angles[2];
        }
}

package Trigonometry;
public class TestTriangle {
        public static void main(String[] args) {
                Triangle A = new Triangle();
                A.FindType();
                System.out.println(A);
        }
```

```
}
```

Comentários a respeito da implementação:

- O programa faz a leitura dos lados do triângulo fornecidos pelo usuário. Em caso de fornecimento de lados inválidos para a formação do triângulo, o programa oferece duas opções:
    1. Permitir ao usuário escolher novos valores;
    2. Decidir por si mesmo novos valores para o triângulo.

        Neste caso, o programa incrementa todos os lados do triângulo até que os mesmos tenham condições de formarem um conjunto de lados válidos.
- O cálculo da área é feito pela fórmula de Herão de Alexandria:

$$A = \sqrt{p(p-a)(p-b)(p-c)},$$

- Os ângulos são descobertos de acordo com a Lei dos Cossenos:

$$a^2 = b^2 + c^2 - 2b \cdot c \cdot cos\widehat{A}$$
$$b^2 = a^2 + c^2 - 2a \cdot c \cdot cos\widehat{B}$$
$$c^2 = a^2 + b^2 - 2a \cdot b \cdot cos\widehat{C}$$

- Foi criado uma função para Truncar os valores de saída dos ângulos e da área