JAVA - AULA 5

= Não tente reinventar a roda. Verifique se já existe o método que deseje
= O nome do método deve significar o que realiza
= Métodos simples (que realizam apenas uma coisa) são mais fáceis de implementar e testar do que complexos (que realizam muitas tarefas)
= main é declarado static para que a JVM não instancie uma classe

Métodos com parâmetros (MaximumFinder.java)

```
import java.util.Scanner;

public class MaximumFinder
{
   public void determineMaximum()
   {
      Scanner input = new Scanner( System.in );

      System.out.print(
         "Digite tres numeros reais separados por espacos: " );
      double number1 = input.nextDouble();
      double number2 = input.nextDouble();
      double number3 = input.nextDouble();

      double result = maximum( number1, number2, number3 );

      System.out.println("O maior e: " + result);
   }

   public double maximum( double x, double y, double z )
   {
      double maximumValue = x;

      if ( y > maximumValue )
         maximumValue = y;

      if ( z > maximumValue )
         maximumValue = z;

      return maximumValue;
   }
}
```

...e chamando seu método (MaximumFinderTest.java)

```
public class MaximumFinderTest
{
   public static void main( String args[] )
   {
      MaximumFinder maximumFinder = new MaximumFinder();
      maximumFinder.determineMaximum();
   }
}
```

= Métodos com muitos parâmetros provavelmente podem ser divididos
= A concatenação de strings pode ser feita com +
= Se y=5, "y + 2 = " + y + 2 imprimirá "y + 2 = 52"
= Se y=5, "y + 2 = " + (y + 2) imprimirá "y + 2 = 7"

Geração de números aleatórios (RollDie.java)

```
import java.util.Random;

public class RollDie
{
   public static void main( String args[] )
   {
      Random randomNumbers = new Random();

      int frequency1 = 0;
      int frequency2 = 0;
      int frequency3 = 0;
      int frequency4 = 0;
      int frequency5 = 0;
      int frequency6 = 0;

      int face;

      for ( int roll = 1; roll <= 6000; roll++ )
      {
         face = 1 + randomNumbers.nextInt( 6 );

         switch (face)
         {
            case 1:
               ++frequency1;
               break;
            case 2:
               ++frequency2;
               break;
            case 3:
               ++frequency3;
               break;
            case 4:
               ++frequency4;
               break;
            case 5:
               ++frequency5;
               break;
            case 6:
               ++frequency6;
               break;
         }
      }

      System.out.println( "Face\tFrequencia" );
      System.out.printf( "1\t%d\n2\t%d\n3\t%d\n4\t%d\n5\t%d\n6\t%d\n",
         frequency1, frequency2, frequency3, frequency4,
         frequency5, frequency6 );
   }
}
```

= nextInt(6) produzirá números aleatórios entre 0 e 5 (1 e 6, somando 1)
= 200 + nextInt(100) produzirá números aleatórios entre 200 e 299
= 2 * nextInt(50) produzirá números aleatórios entre 0 e 98 (pares)

Um jogo de azar com enumerações... (Craps.java)

```
import java.util.Random;

public class Craps
{
   // cria um gerador de números aleatórios para uso no método rollDice
   private Random randomNumbers = new Random();

   // enumeração com constantes que representam o status do jogo
   private enum Status { CONTINUE, WON, LOST };

   // constantes que representam lançamentos comuns dos dados
   private final static int SNAKE_EYES = 2;
   private final static int TREY = 3;
   private final static int SEVEN = 7;
   private final static int YO_LEVEN = 11;
   private final static int BOX_CARS = 12;

   // joga uma partida de craps
   public void play()
   {
      int myPoint = 0; // pontos se não ganhar ou perder na 1a. rolagem
      Status gameStatus; // pode conter CONTINUE, WON ou LOST

      int sumOfDice = rollDice(); // primeira rolagem dos dados

      // determina status do jogo e pontuação baseado no 1o lançamento
      switch ( sumOfDice )
      {
         case SEVEN: // ganha com 7 no primeiro lançamento
         case YO_LEVEN: // ganha com 11 no primeiro lançamento
            gameStatus = Status.WON;
            break;
         case SNAKE_EYES: // perde com 2 no primeiro lançamento
         case TREY: // perde com 3 no primeiro lançamento
         case BOX_CARS: // perde com 12 no primeiro lançamento
            gameStatus = Status.LOST;
            break;
         default: // não ganhou nem perdeu, portanto registra a pontuação
            gameStatus = Status.CONTINUE; // jogo não terminou
            myPoint = sumOfDice; // informa a pontuação
            System.out.printf( "O ponto e %d\n", myPoint );
            break; // opcional no final do switch
      } // switch final

      // enquanto o jogo não estiver completo
      while (gameStatus == Status.CONTINUE) // nem WON nem LOST
      {
         sumOfDice = rollDice(); // lança os dados novamente

         // determina o status do jogo
         if ( sumOfDice == myPoint ) // vitória por pontuação
            gameStatus = Status.WON;
         else
            if ( sumOfDice == SEVEN ) // perde obtendo 7
               gameStatus = Status.LOST;
```

```
      } // fim do while

      // exibe uma mensagem ganhou ou perdeu
      if (gameStatus == Status.WON)
         System.out.println( "Jogador ganha" );
      else
         System.out.println( "Jogador perde" );
   } // fim do método play

   // lança os dados, calcula a soma e exibe os resultados
   public int rollDice()
   {
      // seleciona valores aleatórios do dado
      int die1 = 1 + randomNumbers.nextInt( 6 ); // primeiro lançamento
      int die2 = 1 + randomNumbers.nextInt( 6 ); // segundo lançamento

      int sum = die1 + die2; // soma dos valores dos dados

      // exibe os resultados desse lançamento
      System.out.printf( "Jogador conseguiu %d + %d = %d\n",
         die1, die2, sum );

      return sum; // retorna a soma dos dados
   } // fim do método rollDice
} // fim da classe Craps
```

= Utilize somente maiúsculas para as constantes
= Enumerações podem tornam programas mais fáceis de ler e manter

...e chamando seu método (CrapsTest.java)

```
public class CrapsTest
{
   public static void main( String args[] )
   {
      Craps game = new Craps();
      game.play();
   }
}
```

Escopo das declarações (Scope.java)

```
public class Scope
{
   // atributo acessível para todos os métodos dessa classe
   private int x = 1;

   // método begin cria e inicializa a variável local x
   // e chama os métodos useLocalVariable e useField
   public void begin()
   {
      int x = 5; // variável local x do método sombreia o atributo x

      System.out.printf( "variavel local x no metodo begin e %d\n", x );

      useLocalVariable(); // useLocalVariable tem uma variável local x
```

```
      useField(); // useField utiliza o atributo x da classe Scope
      useLocalVariable(); // useLocalVariable reinicializa variável local
      useField(); // atributo x da classe Scope retém seu valor

      System.out.printf("\nvariavel local x no metodo begin e %d\n", x );
   } // fim do método begin

   // cria e inicializa a variável local x durante cada chamada
   public void useLocalVariable()
   {
      int x = 25; // inicializada toda vez que useLocalVariable é chamado

      System.out.printf( "\nvariavel local x ao entrar no metodo " );
      System.out.printf( "useLocalVariable e %d\n", x );
      ++x; // modifica a variável local x desse método
      System.out.printf( "variavel local x antes de sair do metodo " );
      System.out.printf( "useLocalVariable e %d\n", x );
   } // fim do método useLocalVariable

   // modifica o atributo x da classe Scope durante cada chamada
   public void useField()
   {
      System.out.printf(
         "\natributo x ao entrar no metodo useField e %d\n", x );
      x *= 10; // modifica o atributo x da classe Scope
      System.out.printf(
         "atributo x antes de sair do metodo useField e %d\n", x );
   } // fim do método useField
} // fim da classe Scope
```

...e chamando seu método (ScopeTest.java)

```
public class ScopeTest
{
   public static void main( String args[] )
   {
      Scope testScope = new Scope();
      testScope.begin();
   }
}
```

= Evite trabalhar com variáveis de mesmo nome em escopos diferentes
= Caso precise utilizá-las, tenha muito cuidado

Sobrecarga de método (MethodOverload.java)

```
public class MethodOverload
{
   public void testOverloadedMethods()
   {
      System.out.printf( "Quadrado do inteiro 7 e %d\n", square( 7 ));
      System.out.printf( "Quadrado do real 7.5 e %f\n", square( 7.5 ));
   }

   public int square( int intValue )
   {
```

```
      System.out.printf( "\nMetodo chamado com argumento inteiro: %d\n",
         intValue );
      return intValue * intValue;
   }

   public double square( double doubleValue )
   {
      System.out.printf( "\nMetodo chamado com argumento real: %f\n",
         doubleValue );
      return doubleValue * doubleValue;
   }
}
```

...e chamando seu método (MethodOverloadTest.java)

```
public class MethodOverloadTest
{
   public static void main( String args[] )
   {
      MethodOverload methodOverload = new MethodOverload();
      methodOverload.testOverloadedMethods();
   }
}
```

= CUIDADO com o número, tipo e ordem dos parâmetros

EXERCÍCIOS

1) Forneça o cabeçalho de método para cada um dos seguintes métodos:

a) O método hypotenuse, que aceita dois argumentos de ponto flutuante de precisão dupla side1 e side2 e retorna um resultado de ponto flutuante de dupla precisão.

b) O método smallest, que recebe três inteiros x, y e z e retorna um inteiro.

c) O método instructions, que não aceita argumentos e não retorna um valor.

d) O método intToFloat, que aceita um argumento inteiro number e retorna um resultado de ponto flutuante.

2) Escreva um aplicativo completo para solicitar ao usuário o raio (do tipo doublé) de uma esfera e chame o método sphereVolume para calcular e exibir o volume da esfera. Utilize a seguinte instrução para calcular o volume:

```
double volume = ( 4.0 / 3.0 ) * Math.PI * Math.pow( raio, 3)
```