CS 152
Computer Programming Fundamentals
Project 4 – Method Practice Solution

Brooke Chenoweth

University of New Mexico

Spring 2024

minOfThree

```
/**
 * Returns smallest of its arguments.
 * @param x First argument
 * @param y Second argument
 * @param z Third argument
 * @return Minimum of x, y and z
 */
public static int minOfThree( int x, int y, int z) {
  if(x < y) {
    if(x < z) return x;
    else return z;
  } else {
    if(y < z) return y;
    else return z;
  }
}
```

minOfThree – local variable

```
/**
 * Returns smallest of its arguments.
 * @param x First argument
 * @param y Second argument
 * @param z Third argument
 * @return Minimum of x, y and z
 */
public static int minOfThree( int x, int y, int z) {
  int smallest = x;
  if(y < smallest) {
    smallest = y;
  }
  if(z < smallest) {
    smallest = z;
  }
  return smallest;
```

minOfThree – ternary operator

```
/**
 * Returns smallest of its arguments.
 * @param x First argument
 * @param y Second argument
 * @param z Third argument
 * @return Minimum of x, y and z
 */
public static int minOfThree( int x, int y, int z) {
      return
          x < y ? (x < z ? x : z) : (y < z ? y : z);

        // like an if statement, but is an expression
        // BooleanExpr ? ValueIfTrue : ValueIfFalse
 }
```

minOfThree – sorting

```
/**
 * Returns smallest of its arguments.
 * @param x First argument
 * @param y Second argument
 * @param z Third argument
 * @return Minimum of x, y and z
 */
public static int minOfThree( int x, int y, int z) {
  // This is really overkill for just 3 values
  int[] nums = {x, y, z};
  java.util.Arrays.sort(nums);
  return nums[0];
}
```

isVowel

```
/**
 * Is the character given a vowel?
 * A vowel is one of a, e, i, o, or u (upper or lower
 * @param c Character to check
 * @return True if c is a vowel, false if not
 */
public static boolean isVowel(char c) {
  return c == 'a' || c == 'e' || c == 'i'
      || c == 'o' || c == 'u'
      || c == 'A' || c == 'E' || c == 'I'
      || c == 'O' || c == 'U';
}
```

Returning boolean expressions

If you have code of the form

```
if(booleanExpr) {
  return true;
} else {
  return false;
}
```

you can write it more concisely as

```
return booleanExpr;
```

isVowel – switch, toLowerCase

```
/**
 * Is the character given a vowel?
 * A vowel is one of a, e, i, o, or u (upper or lower
 * @param c Character to check
 * @return True if c is a vowel, false if not
 */
public static boolean isVowel(char c) {
    switch(Character.toLowerCase(c)) {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        return true;
    default:
        return false;
    }
}
```

isVowel – String method

```
/**
 * Is the character given a vowel?
 * A vowel is one of a, e, i, o, or u (upper or lower
 * @param c Character to check
 * @return True if c is a vowel, false if not
 */
public static boolean isVowel(char c) {
  return "aeiouAEIOU".indexOf(c) >= 0;
}
```

hasVowel

```
/**
 * Does the given string contain a vowel?
 * @param s String to check
 * @return True if s contains a vowel, false if not
 */
public static boolean hasVowel(String s) {
    for(int i = 0; i < s.length(); i++) {
        if(isVowel(s.charAt(i))) {
            return true;
        }
    }
    return false;
}
```

switchCase – char math

```java
/**
 * Flip character's case between upper and lower.
 * If lowercase, return uppercase equivalent.
 * If uppercase, return lowercase equivalent.
 * If not a letter, return unchanged.
 * @param c Character to process
 * @return Character with case switched.
 */
public static char switchCase(char c) {
  int offset = 'A' - 'a';
  if('a' <= c && c <= 'z') {
    return (char)(c + offset);
  } else if('A' <= c && c <= 'Z') {
    return (char)(c - offset);
  } else {
    return c;
  }
}
```

switchCase – Character methods

```java
/**
 * Flip character's case between upper and lower.
 * If lowercase, return uppercase equivalent.
 * If uppercase, return lowercase equivalent.
 * If not a letter, return unchanged.
 * @param c Character to process
 * @return Character with case switched.
 */
public static char switchCase(char c) {
  if(Character.isLowerCase(c)) {
    return Character.toUpperCase(c);
  } else if(Character.isUpperCase(c)) {
    return Character.toLowerCase(c);
  } else {
    return c;
  }
}
```

toOtherCase

```
/**
 * Return a string with each lowercase letter
 * converted to uppercase, each uppercase letter
 * switched to lowercase, and each non-letter
 * character left unchanged
 * @param s The string to process
 * @return New string with upper and lower case switc
 */
public static String toOtherCase(String s) {
    String result = "";
    for(int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        result += switchCase(c);
    }
    return result;
}
```

averageOddNumbers – the long way

```
public static double averageOddNumbers(
    int a, int b, int c, int d, int e ) {
  int count = 0;
  double sum = 0.0;

  if(a % 2 != 0) {
    count++;
    sum += a;
  }
  if(b % 2 != 0) {
    count++;
    sum += b;
  }
  if(c % 2 != 0) {
    count++;
    sum += c;
  }
  if(d % 2 != 0) {
    count++;
    sum += d;
  }
  if(e % 2 != 0) {
    count++;
    sum += e;
  }

  if(count == 0) {
    return -1000;
  } else {
    return sum/count;
  }
}
```

averageOddNumbers – array, helper

```
public static double averageOddNumbers(
    int a, int b, int c, int d, int e ) {
  int[] arr = {a, b, c, d, e};
  return averageOddArray(arr);
}

public static double averageOddArray(int[] vals) {
  double sum = 0;
  int count = 0;

  for(int i = 0; i < vals.length; i++) {
    if(vals[i] % 2 != 0) {
      sum += vals[i];
      count++;
    }
  }

  if(count == 0) return -1000;
  else return sum / count;
}
```

averageOddNumbers – special for loop

```
public static double averageOddNumbers(
    int a, int b, int c, int d, int e ) {
  int[] arr = {a, b, c, d, e};
  return averageOddArray(arr);
}

public static double averageOddArray(int[] vals) {
  double sum = 0;
  int count = 0;

  for(int val : vals) {
    if(val % 2 != 0) {
      sum += val;
      count++;
    }
  }

  if(count == 0) return -1000;
  else return sum / count;
}
```

countVowels

```
/**
 * Returns the number of vowels in the given string.
 * @param s String to check
 * @return Number of vowels in the string.
 */
public static int countVowels(String s) {
    int count = 0;
    for(int i = 0; i < s.length(); i++) {
        if(isVowel(s.charAt(i))) {
            count++;
        }
    }
    return count;
}
```

hasVowel – using countVowels

```
/**
 * Does the given string contain a vowel?
 * @param s String to check
 * @return True if s contains a vowel, false if not
 */
public static boolean hasVowel(String s) {
  return countVowels(s) > 0;
}
```

totalMealPrice

```
/**
 * Calculates meal total after adding a tip.
 * @param meal Cost of meal. Must be positive
 * @param tip Tip percentage
 *            (must be between 0 and .9)
 * @return Total bill amount after adding tip.
 */
public static double totalMealPrice(int meal,
                                    double tip) {

  if(meal <= 0 || tip < 0 || tip > 0.9) return -1;

  return meal * (1 + tip);
}
```

totalMealPrice – another version

```
/**
 * Calculates meal total after adding a tip.
 * @param meal Cost of meal. Must be positive
 * @param tip Tip percentage
 *            (must be between 0 and .9)
 * @return Total bill amount after adding tip.
 */
public static double totalMealPrice(int meal,
                                    double tip) {

  if(meal > 0 && tip >= 0 && tip <= 0.9) {
    double tipVal = meal*tip;
    double total = meal + tipVal;
    return total;
  } else {
    return -1;
  }
}
```