COMP0004 Object-Oriented Programming
Exercises 1
Example Answers

Core questions

Q1.1 Write a program that inputs a sequence of Strings until the word **stop** is entered.

Hint: You compare strings using compareTo, as outlined in the preceding notes.

Answer:

The answer uses a do-while loop, where the loop body is evaluated at least once. This is an example of where the do-while loop is a better solution than a while or for loop, as the loop body always needs to be evaluated at least once and the boolean condition evaluated after an attempt at input has been made. It also avoids needing to duplicate the prompt message and input code before the loop body.

```
public class Q1
{
  public void inputUntilStopIsTyped()
  {
    Input input = new Input();
    String in;
    do
    {
      System.out.print
        ("Enter some text, type 'stop' to exit the program: ");
      in = input.nextLine();
      System.out.println("You typed: " + in);
    } while (in.compareTo("stop") != 0);
  }

  public static void main(String[] args)
  {
    new Q1().inputUntilStopIsTyped();
  }
}
```

The boolean expression could also be written as !in.equals(“stop”)

Q1.2 Write a program to input 10 doubles (values of type double) and store them in an array. Then compute and display the average of the values input. Try entering negative as well as positive numbers. Consider writing one method to do the input and another to compute the average.

Answer:

```
public class Q2
{
  public double[] inputDoubles()
  {
```

```
    Input input = new Input();
    double[] values = new double[10];

    for (int n = 0 ; n < values.length ; n++)
    {
      System.out.print("Enter a double: ");
      values[n] = input.nextDouble();
    }
    return values;
  }

  public double average(double[] values)
  {
    double sum = 0.0;
    for (double value : values)
    {
      sum += value;
    }
    return sum/values.length;
  }

  public void go()
  {
    double average = average(inputDoubles());
    System.out.println("The average is: " + average);
  }

  public static void main(String[] args)
  {
    new Q2().go();
  }
}
```

This works but the program is terminated whenever you make a mistake typing in a double number (this is deliberate behaviour in the Input class). Below is another version that handles input errors without terminating the program and also uses an instance variable to store the array rather than passing the array as a parameter to methods.

```
public class Q2a
{
  private Input input;
  private double[] values;

  public Q2a()
  {
    input = new Input();
    values = new double[10];
  }

  public double inputDouble()
  {
    while (true)
    {
      System.out.print("Enter a double value: ");
      if (input.hasNextDouble())
      {
         return input.nextDouble();
```

```
      }
      input.nextLine();
      System.out.println("Not a double value, please try again.");
    }
  }

  public void inputArrayOfDoubles()
  {
    for (int n = 0 ; n < values.length ; n++)
    {
      values[n] = inputDouble();
    }
  }

  public double average()
  {
    double sum = 0.0;
    for (double value : values)
    {
      sum += value;
    }
    return sum/values.length;
  }

  public void go()
  {
    inputArrayOfDoubles();
    double average = average();
    System.out.println("The average is: " + average);
  }

  public static void main(String[] args)
  {
    new Q2a().go();
  }
}
```

A further variation of the inputDouble method is shown below. This time it reads the double as a String value and then attempts to convert it to an actual value of type double using the library method parseDouble. This illustrates the use of the Java exception handling mechanism, as an exception will be thrown by the parseDouble method if it cannot convert the String to a double.

```
public double inputDouble()
  {
    while (true)
    {
      System.out.print("Enter a double value: ");
      String s = input.nextLine();
      try
      {
        return Double.parseDouble(s);
      }
      catch (NumberFormatException e)
      {
        System.out.println("Not a double value, please try again.");
      }
```

```
    }
  }
```

Q1.3 Write a program to input 10 words, storing them as strings in an ArrayList<String> (not an array), and then display the words in reverse sorted order.

Answer:

```
import java.util.ArrayList;
import java.util.Collections;

public class Q3
{
  public  ArrayList<String> inputWords()
  {
    Input input = new Input();
    ArrayList<String> words = new ArrayList<String>();

    for (int n = 0 ; n < 10 ; n++)
    {
      System.out.print("Enter a word: ");
      words.add(input.nextLine());
    }
    return words;
  }

  public void sort(ArrayList<String> words)
  {
    Collections.sort(words, Collections.reverseOrder());
  }

  public void go()
  {
    ArrayList<String> words = inputWords();
    sort(words);
    System.out.println(words);
  }

  public static void main(String[] args)
  {
    new Q3().go();
  }
}
```

Class Collections provides a number of utility methods providing operations such as sorting. Knowing what is in the class library can make writing code a lot easier!

You might wonder what Collections.reverseOrder() actually is. One way to find out, of course(!), is to use the Java Documentation (JavaDoc). This will tell you that reverseOrder is a static (class) method that returns a Comparator object implementing the Comparable interface. A comparator is used to compare two values based on their defined ordering, and is actually an object wrapping a comparison method. Collections.sort uses a comparator object to compare values when sorting rather than having a fixed built in way of comparing values, which would only work for one type of value. By sorting using a comparator a single sort implementation can sort values of any type if given a suitable comparator, with the constraint that all values being sorted have to have the same type (or be comparable in a systematic way).

Q1.4 Write a program to generate 10000 random doubles between -0.9999999 and +0.9999999. Print out the largest, smallest and the average values.

Do you need an array? Think very carefully about this.

Hint: Look at the documentation for class Random and find the method nextDouble.
Another hint: Negative values?? class Random also has nextBoolean...

Answer:

```
import java.util.Random;

public class Q4
{
  private static final int RANDOM_COUNT = 10000;
  private Random generator;
  private double sum;
  private double minimum;
  private double maximum;

  public Q4()
  {
    generator = new Random();
    sum = 0.0;
    minimum = 0.0;
    maximum = 0.0;
  }

  private int sign()
  {
    return generator.nextBoolean() ? 1 : -1;
  }

  private double getDouble()
  {
    return generator.nextDouble() * sign();
  }

  private void generateDoubles(int count)
  {
    for (int counter = 0; counter < count; counter++)
    {
      double number = getDouble();
      sum += number;
      minimum = Math.min(minimum,number);
      maximum = Math.max(maximum,number);
    }
  }

  private void printResults()
  {
    System.out.println("Average: " + sum / RANDOM_COUNT);
    System.out.println("Min: " + minimum);
    System.out.println("Max: " + maximum);
  }
```

```
  public void go()
  {
    generateDoubles(RANDOM_COUNT);
    printResults();
  }

  public static void main(String[] args)
  {
    new Q4().go();
  }
}
```

If you run this program it prints out results like this:

```
Average: -0.004340520616034457
Min: -0.9999134450769643
Max: 0.9999662613766744
```

but how reliable are all those digits after the decimal point? It would be better restrict the output to a fixed number of digits. This can be done using the printf (print formatted) method like this:

```
  private void printResults()
  {
    System.out.printf("Average: %.4f\n", sum / RANDOM_COUNT);
    System.out.printf("Min: %.4f\n", minimum);
    System.out.printf("Max: %.4f\n", maximum);
  }
```

which displays:

```
Average: -0.0031
Min: -1.0000
Max: 0.9996
```

The pattern “%.4f” specifies how to print the number supplied by the second parameter. The “.4” means four digits after the decimal point and the “f” means print a decimal floating point number. The printf method in Java is an implementation of the printf function from the C programming language. This style of formatting is widely used on the Unix platform and supported by quite a few programming languages.

Q1.5 Write a one-class program that has the following methods:

- a method that inputs and returns a double value,
- a method that takes two double parameters, adds them together and returns the square root of the result,
- a main method to create an object and call the other two methods, displaying the result of calling the second method.

Answer:

```
public class Q5
{
  public double inputDouble(Input input)
  {
    while (true)
    {
      System.out.print("Enter a double value: ");
```

```
      String s = input.nextLine();
      try
      {
        return Double.parseDouble(s);
      }
      catch (NumberFormatException e)
      {
        System.out.println("Not a double value, please try again.");
      }
    }
  }

  public double calculate(double first, double second)
  {
    return Math.sqrt(first + second);
  }

  public static void main(String[] args)
  {
    Input input = new Input();
    Q5 q5 = new Q5();
    double first = q5.inputDouble(input);
    double second = q5.inputDouble(input);
    System.out.println("Result: " + q5.calculate(first,second));
  }
}
```

Note that in all the programs getting input from the keyboard, only one Input object is created and reused where needed. Creating multiple Input objects, all trying to get input from the same keyboard, will not work properly due to the buffering of input.

Q1.6 Repeat Q1.5 but store the double values in two instance variables rather than using parameters.

Hint: you will need another instance method to call the input method to get values to store in each instance variable and to call the other methods. This instance method should be public, the others private. The input method should be unchanged.

Answer:

```
public class Q6
{
  private double first;
  private double second;
  Input input;

  public Q6()
  {
    input = new Input();
  }

  private double inputDouble()
  {
    while (true)
    {
```

```
      System.out.print("Enter a double value: ");
      String s = input.nextLine();
      try
      {
        return Double.parseDouble(s);
      }
      catch (NumberFormatException e)
      {
        System.out.println("Not a double value, please try again.");
      }
    }
  }

  private double calculate()
  {
    return Math.sqrt(first + second);
  }

  public void go()
  {
    first = inputDouble();
    second = inputDouble();
    System.out.println("Result: " + calculate());
  }

  public static void main(String[] args)
  {
    new Q6().go();
  }
}
```

Q1.7 Write a method: public String toBase(int n, int b) that converts n to its String representation in number base b, i.e., toBase(3,2) will return "11". Put the method into a one-class program that allows you to input values and use the method.

Answer:

```
public class Q7
{
  private Input input;

  public Q7()
  {
    input = new Input();
  }

  private int inputInteger(String prompt)
  {
    while (true)
    {
      System.out.print(prompt);
      String s = input.nextLine();
      try
      {
```

```
        return Integer.parseInt(s);
      }
      catch (NumberFormatException e)
      {
        System.out.println("Not a double value, please try again.");
      }
    }
  }

  private String toBase(int number, int base)
  {
    return Integer.toString(number,base);
  }

  private void go()
  {
    int number = inputInteger("Enter number to convert: ");
    int base = inputInteger("Enter number base to convert to: ");
    System.out.println("Answer: " + toBase(number,base));
  }

  public static void main(String[] args)
  {
    new Q7().go();
  }
}
```

The key here is to make good use of the JavaDoc documentation and discover that someone has already done the hard work in the form of the toString method of class Integer.

Q1.8 Write methods to do the following:

- Convert from millimetres to feet.
- Convert from metres to inches.
- Convert from kilometres to yards.

Include the methods in an interactive program that lets the user select which conversion to perform, and then inputs a value and displays the result.

An interactive program means one that displays a simple text menu, such as:

```
1. Convert millimetres to feet.
2. Convert metres to inches.
3. Convert kilometres to yards.
4. Quit
```

and then asks the user to type in a value from 1 to 4 to select which action to take. A loop will continually redisplay the menu until the user selects Quit (number 4) to terminate the program.

Notes: The conversion methods should take the value to be converted as a parameter and return a result. They should not do any input or display the result. Add additional methods if they will help structure the program.

Answer:

```
public class Q8
{
```

```
private Input input;

public Q8()
{
  input = new Input();
}

private double inputDouble(String prompt)
{
  while (true)
  {
    System.out.print(prompt);
    String s = input.nextLine();
    try
    {
      return Double.parseDouble(s);
    }
    catch (NumberFormatException e)
    {
      System.out.println(
        "The input could not be recognised, please try again.");
    }
  }
}

private int inputInteger(String prompt)
{
  while (true)
  {
    System.out.print(prompt);
    String s = input.nextLine();
    try
    {
      return Integer.parseInt(s);
    }
    catch (NumberFormatException e)
    {
      System.out.println(
        "The input could not be recognised, please try again.");
    }
  }
}

private double millimetresToFeet(double millimetres)
{
  return millimetres * 0.00328;
}

private double metresToInches(double metres)
{
  return metres * 39.37;
}

private double kilometresToYards(double kilometres)
{
  return kilometres * 1093.613;
```

```
}

private void convert(int selection)
{
  double value = inputDouble("Enter value to convert: ");
  double result = doConversion(selection, value);
  System.out.println("Converted value: " + result);
}

private double doConversion(int selection, double value)
{
  double result = 0.0;
  switch (selection)
  {
    case 1:
      result = millimetresToFeet(value);
      break;
    case 2:
      result = metresToInches(value);
      break;
    case 3:
      result = kilometresToYards(value);
      break;
  }
  return result;
}

private void displayMenu()
{
  System.out.println("Conversion Program");
  System.out.println("1 -- Millimetres to Feet");
  System.out.println("2 -- Metres to Inches");
  System.out.println("3 -- Kilometres to Yards");
  System.out.println("4 -- Quit");
}

private void go()
{
  while (true)
  {
    displayMenu();
    int selection = inputInteger("Select Option: ");
    if ((selection < 1) || (selection > 4))
    {
      System.out.println("Invalid selection, try again");
      continue;
    }
    if (selection == 4)
    {
      System.out.println("Goodbye");
      break;
    }
    convert(selection);
  }
}
```

```
  public static void main(String[] args)
  {
    new Q8().go();
  }
}
```

Q1.9 Write a one-class program to determine if a long integer is a palindrome (i.e., represents the same value when reversed, for example 123454321).

Answer:

```
public class Q9
{
  private Input input;

  public Q9()
  {
    input = new Input();
  }

  private long inputLong(String prompt)
  {
    while (true)
    {
      System.out.print(prompt);
      String s = input.nextLine();
      try
      {
        return Long.parseLong(s);
      }
      catch (NumberFormatException e)
      {
        System.out.println(
          "The input could not be recognised, please try again.");
      }
    }
  }

  private boolean isPalindrome(long n)
  {
    String number = "" + n;
    return number.equals(
      new StringBuilder(number).reverse().toString());
  }


  private void go()
  {
    long number = inputLong("Enter number to check: ");
    if (isPalindrome(number))
    {
      System.out.println("Is a palindrome");
    }
    else
    {
```

```
      System.out.println("Is not a palindrome");
    }
  }

  public static void main(String[] args)
  {
    new Q9().go();
  }
}
```

Q1.10 The example palindrome program seen in the preceding notes includes a tidyString method to remove spaces and make all characters lower case before checking whether a sentence is a palindrome. However, many example palindromes also include punctuation and spaces, for example “Neil, a trap! Sid is part alien!”. Modify the example palindrome program to remove the punctuation and spaces and compare any string.

Answer:

```
public class Q10
{
  private String reverse(String s)
  {
    return new StringBuilder(s).reverse().toString();
  }

  private boolean check(String s1, String s2)
  {
    String s = reverse(s2);
    return s1.equals(s);
  }

  private String getInput()
  {
    Input in = new Input();
    System.out.print("Enter text to check: ");
    return in.nextLine();
  }

  private String tidyString(String s)
  {
    return s.toLowerCase().replaceAll("[^a-z]", "");
  }

  public void testForPalindrome(String s)
  {
    if (check(tidyString(s), tidyString(s)))
    {
      System.out.println("Text is a palindrome.");
    }
    else
    {
      System.out.println("Text is not a palindrome.");
    }
  }

  public void go()
```

```
  {
    testForPalindrome(getInput());
  }

  public static void main(final String[] args)
  {
    new Q10().go();
  }
}
```

Think Regular Expressions and this turns out to be easy! The tidyString method body is changed to this:

```
return s.toLowerCase().replaceAll("[^a-z]", "");
```

Change all uppercase characters in the string to lowercase and then replace everything that is not a character in the range a-z with an empty string, resulting in the unwanted characters being removed. The [^a-z] is a regular expression meaning a character not in the range a-z.

Q1.11 Write a one-class program with suitable methods to read a text file character by character and count how frequently each character occurs.

Hints: You will need to use the FileInput class, which is used in a similar way to the Input class. See the notes on using FileInput.

The FileInput .class file needs to be in the same directory as the program you are working on. Copy FileInput.class to all directories containing programs that need to use it.

Answer:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class Q11
{
  private Map<Character,Integer> characterCounts;
  private Input input;

  public Q11()
  {
    characterCounts = new HashMap<Character,Integer>();
    input = new Input();
  }

  private void countChar(char c)
  {
    if (characterCounts.containsKey(c)){
      characterCounts.put(c,characterCounts.get(c) + 1);
    }
    else
    {
      characterCounts.put(c,1);
    }
  }

  private void readFile(String name)
```

```
  {
    FileInput file = new FileInput(name);
    while (file.hasNextChar())
    {
      char c = file.nextChar();
      countChar(c);
    }
    file.close();
  }

  private String getFileName(String message)
  {
    System.out.print(message);
    return input.nextLine();
  }

  private void printResult()
  {
    ArrayList<Character> characters =
      new ArrayList<Character>(characterCounts.keySet());
    Collections.sort(characters);
    for (Character c : characters)
    {
      System.out.println(c +
                         " (Unicode value " + (int)c.charValue() +
                         "): " + characterCounts.get(c));
    }
  }

  private void go()
  {
    String fileName = getFileName("Enter name of text file: ");
    readFile(fileName);
    printResult();
  }

  public static void main(String[] args)
  {
    new Q11().go();
  }
}
```

A map (hash table) provides an ideal data structure for storing the character counts. Once the counting has been done, the map can be asked for its key values, which will be returned as a set. As it is useful to print out the results in sorted order, an ArrayList is created from the set of key values and then sorted. The sorted keys are used to access the values in the hash table.

Q1.12 Write a program using methods that copies and reverses a text file (i.e., so that the destination file contains a backwards copy of the original file contents).

This program will need both the FileInput and FileOutput classes (described in separate notes).

Answer:

```
import java.util.ArrayList;
```

```
import java.util.Collections;

public class Q12
{
  private Input input;
  private ArrayList<String> fileContent;

  public Q12()
  {
    input = new Input();
    fileContent = new ArrayList<String>();
  }

  public String getFileName(String message)
  {
    System.out.print(message);
    return input.nextLine();
  }

  public void readFile(String name)
  {
    FileInput fileInput = new FileInput(name);
    while (fileInput.hasNextLine())
    {
      fileContent.add(fileInput.nextLine());
    }
    fileInput.close();
  }

  public String reverseString(String s)
  {
    return new StringBuilder(s).reverse().toString();
  }

  public void reverseEachLine()
  {
    for (int line = 0 ; line < fileContent.size() ; line++)
    {
      fileContent.set(line,reverseString(fileContent.get(line)));
    }
  }

  public void reverseOrderOfLines()
  {
    Collections.reverse(fileContent);
  }

  public void reverseFileContent()
  {
    reverseOrderOfLines();
    reverseEachLine();
  }

  public void writeFile(String name)
  {
    FileOutput output = new FileOutput(name);
```

```
    for (String line : fileContent)
    {
      output.writeString(line);
      output.writeEndOfLine();
    }
    output.close();
  }

  public void go()
  {
    String inputFile = getFileName("Enter name of file to reverse: ");
    String outputFile =
      getFileName("Enter name of file to write reverse file to: ");

    readFile(inputFile);
    reverseFileContent();
    writeFile(outputFile);
  }

  public static void main(String arg[])
  {
    new Q12().go();
  }
}
```

This program works by first reading each line of the file into an ArrayList of Strings, then reversing the order the lines and finally reversing the characters in each line. Reversing the characters in a String exploits the fact that a reverse method is available in the Java class libraries for class StringBuilder (a class used to build strings out of other strings and characters).

Q1.13 Further modify the palindrome program to read a file containing strings (one per line) and checking whether each string is a palindrome. The palindromes found should be written to a separate output file. Note, do a Google search to find many example palindromes.

Answer:

```
public class Q13
{
  private static final String COMMENT_MARKER = "#";

  private String reverse(String s)
  {
    return new StringBuilder(s).reverse().toString();
  }

  private boolean isReverseOf(String s1, String s2)
  {
    String s = reverse(s2);
    return s1.equals(s);
  }

  private String tidyString(String s)
  {
    return s.toLowerCase().replaceAll("[^a-z]", "");
  }
```

```
  private boolean isPalindrome(String s)
  {
    return isReverseOf(tidyString(s), tidyString(s));
  }

  private String inputString(Input input, String message)
  {
    System.out.print(message);
    return input.nextLine();
  }

  private void processFile(String inFile, String outFile)
  {
    FileInput fileInput = new FileInput(inFile);
    FileOutput fileOutput = new FileOutput(outFile);
    checkEachLine(fileInput, fileOutput);
    fileOutput.close();
    fileInput.close();
  }

  private void checkEachLine(FileInput fileInput, FileOutput fileOutput)
  {
    while (fileInput.hasNextLine())
    {
      String line = fileInput.nextLine();

      // Ignore a comment line in the input file
      if (line.startsWith(COMMENT_MARKER)) { continue; }

      if (isPalindrome(line))
      {
        writePalindromeToFile(fileOutput, line);
      }
    }
  }

  private void writePalindromeToFile(FileOutput fileOutput, String line)
  {
    fileOutput.writeString(line);
    fileOutput.writeEndOfLine();
  }

  public void go()
  {
    Input input = new Input();
    String inFile = inputString(input, "Enter input file name:");
    String outFile = inputString(input, "Enter output file name:");
    processFile(inFile,outFile);
  }

  public static void main(final String[] args)
  {
    new Q13().go();
  }
}
```

And here is a file containing palindromes, one per line:

```
# Data for the Q13 palindrome testing program.
# Each line of text is to be tested to see if it is a palindrome
# except comment lines starting with #.
# These examples come from Wikipedia
# http://en.wikipedia.org/wiki/Palindrome
A dog! A panic in a pagoda!
Ah, Satan sees Natasha.
Are we not drawn onward, we few, drawn onward to new era?
Do geese see God?
I prefer pi.
If I had a hi-fi.
Ma is as selfless as I am.
Mr. Owl ate my metal worm.
Never odd or even.
No devil lived on.
No, sir, away! A papaya war is on!
Red rum, sir, is murder.
Rise to vote, sir.
So many dynamos!
Satan! Oscillate my metallic sonatas.
# These are not palindromes.
Not a palindrome.
Just some words.
# These palindromes are from
# http://www.fun-with-words.com/palin_example.html
Don't nod
Dogma: I am God
Never odd or even
Too bad — I hid a boot
Rats live on no evil star
No trace; not one carton
Was it Eliot's toilet I saw?
Murder for a jar of red rum
May a moody baby doom a yam?
Go hang a salami; I'm a lasagna hog!
Satan, oscillate my metallic sonatas!
A Toyota! Race fast... safe car: a Toyota
Straw? No, too stupid a fad; I put soot on warts
Are we not drawn onward, we few, drawn onward to new era?
Doc Note: I dissent. A fast never prevents a fatness. I diet on cod
No, it never propagates if I set a gap or prevention
Anne, I vote more cars race Rome to Vienna
Sums are not set as a test on Erasmus
Kay, a red nude, peeped under a yak
Some men interpret nine memos
Campus Motto: Bottoms up, Mac
Go deliver a dare, vile dog!
Madam, in Eden I'm Adam
Oozy rat in a sanitary zoo
Ah, Satan sees Natasha
Lisa Bonet ate no basil
God saw I was dog
Dennis sinned
```

Q1.14 Write a program using methods to display your name, or any other message, in the *middle* of a line 80 characters wide.

Answer:

```
public class Q14
{
  private String getInput()
  {
    System.out.print("Enter the string to display: ");
    return new Input().nextLine();
  }

  private void writeLine(String message)
  {
    for (int i = 0; i < (80 - message.length()) / 2; i++)
    {
      System.out.print(' ');
    }
    System.out.println(message);
  }

  public static void main(String[] args)
  {
    Q14 application = new Q14();
    application.writeLine(application.getInput());
  }
}
```

This has to be done by working out how many spaces to print before displaying the message.

Q1.15 Write a program that uses a *recursive* method to calculate the product of a sequence of numbers specified by the user. For example, if the user specifies 4 to 8, the method calculates 4*5*6*7*8. Any range can be used, including the use of negative numbers, and the program must correctly determine the values in the range.

Answer:

```
public class Q15
{
  private int inputInteger(Input input, String prompt)
  {
    while (true)
    {
      System.out.print(prompt);
      String s = input.nextLine();
      try
      {
        return Integer.parseInt(s);
      }
      catch (NumberFormatException e)
      {
        System.out.println("Not an integer, please try again.");
      }
    }
  }
```

```
  private int product(int start, int end)
  {
    if (start > end) { return 1; }
    return start * product(start+1,end);
  }

  public void go()
  {
    Input input = new Input();
    int start =
      inputInteger(input, "Enter integer at start of the range:");
    int end = inputInteger(input, "Enter integer at end of the range:");
    int result = product(Math.min(start,end),Math.max(start,end));
    System.out.println("The product is: " + result);
  }

  public static void main(final String[] args)
  {
    new Q15().go();
  }
}
```

Note that any range that includes zero such as -1..3 will, of course, have a product of zero.

Q1.16 Write a method that takes two character array parameters and returns true if both arrays contain the same characters but not necessarily in the same order. Note, character arrays are of type char[].

Answer:

This is about writing a method to determine if one array is a permutation of the other. The first answer does this by iterating through the arrays counting characters.

```
public class Q16
{
  private boolean isPermutation(char[] a, char[] b)
  {
    if (a.length != b.length)
    {
      return false;
    }
    for (int i = 0; i < a.length; ++i)
    {
      int countA = 0;
      int countB = 0;
      char c = a[i];
      for (int j = 0; j < a.length; ++j)
      {
        if (c == a[j])
        {
          countA++;
        }
        if (c == b[j])
        {
          countB++;
        }
      }
```

```
      if (countA != countB)
      {
        return false;
      }
    }
    return true;
  }

  private void assertTrue(boolean b, String message)
  {
    if (!b)
    {
      System.out.println("Test failed: " + message);
    }
  }

  private void runTest()
  {
    char[] a = {'a','b','c','d','e','f'};
    char[] b = {'a','b','c','d','e','f'};//{'a','b','c','d'};
    assertTrue(isPermutation(a, a), "a is not a permutation of a");
    assertTrue(isPermutation(b, b), "b is not a permutation of b");
    assertTrue(!isPermutation(a, b), "a is not different from b");
    assertTrue(!isPermutation(b, a), "b is not different from a");
    char[] c = {'f','e','d','c','b','a'};
    assertTrue(isPermutation(c, c), "c is not a permutation of c");
    assertTrue(isPermutation(a, c), "a is not a permutation of c");
    assertTrue(isPermutation(c, a), "c is not a permutation of a");
    char[] d = {'a','a','a','b'};
    char[] e = {'a','b','b','b'};
    assertTrue(isPermutation(d, d), "d is not a permutation of d");
    assertTrue(isPermutation(e, e), "e is not a permutation of e");
    assertTrue(!isPermutation(d, e), "d is not different from e");
    assertTrue(!isPermutation(e, d), "e is not different from d");
    char[] f = {'a'};
    assertTrue(isPermutation(f, f), "f is not a permutation of f");
    assertTrue(!isPermutation(f, e), "f is not different from e");
    char[] g = {};
    assertTrue(isPermutation(g, g), "g is not a permutation of g");
    assertTrue(!isPermutation(g, f), "g is not different from f");
    assertTrue(!isPermutation(g, e), "g is not different from e");
  }

  public static void main(final String[] args)
  {
    (new Q16()).runTest();
  }
}
```

As we want evidence that the permutation method works, some simple test code is provided to assert that the method returns the correct result.

An alternative way of testing for a permutation is to sort the two arrays being compared and then checking if they are equal; the same sequence of characters.

```
import java.util.Arrays;
```

```
public class Q16a
{
  private boolean isPermutation(char[] a, char[] b)
  {
    if (a.length != b.length)
    {
      return false;
    }
    char[] aa = a.clone();
    Arrays.sort(aa);
    char[] bb = b.clone();
    Arrays.sort(bb);
    return Arrays.equals(aa, bb);
  }

  private void assertTrue(boolean b, String message)
  {
    if (!b)
    {
      System.out.println("Test failed: " + message);
    }
  }

  private void runTest()
  {
    char[] a = {'a','b','c','d','e','f'};
    char[] b = {'a','b','c','d'};
    assertTrue(isPermutation(a, a), "a is not a permutation of a");
    assertTrue(isPermutation(b, b), "b is not a permutation of b");
    assertTrue(!isPermutation(a, b), "a is not different from b");
    assertTrue(!isPermutation(b, a), "b is not different from a");
    char[] c = {'f','e','d','c','b','a'};
    assertTrue(isPermutation(c, c), "c is not a permutation of c");
    assertTrue(isPermutation(a, c), "a is not a permutation of c");
    assertTrue(isPermutation(c, a), "c is not a permutation of a");
    char[] d = {'a','a','a','b'};
    char[] e = {'a','b','b','b'};
    assertTrue(isPermutation(d, d), "d is not a permutation of d");
    assertTrue(isPermutation(e, e), "e is not a permutation of e");
    assertTrue(!isPermutation(d, e), "d is not different from e");
    assertTrue(!isPermutation(e, d), "e is not different from d");
    char[] f = {'a'};
    assertTrue(isPermutation(f, f), "f is not a permutation of f");
    assertTrue(!isPermutation(f, e), "f is not different from e");
    char[] g = {};
    assertTrue(isPermutation(g, g), "g is not a permutation of g");
    assertTrue(!isPermutation(g, f), "g is not different from f");
    assertTrue(!isPermutation(g, e), "g is not different from e");
  }

  public static void main(final String[] args)
  {
    new Q16a().runTest();
  }
}
```

Note that the arrays are cloned (copied) before they are sorted, so that the original arrays remain unchanged. Remember that arrays are represented by objects and passing an array parameter to a method is actually passing a reference to the array object.

Harder Questions

Q1.17

a) Write a method to test if an integer of type long is a prime number. The method should return a boolean value. Test your method by writing a test one-class program that reads an integer typed at the keyboard and states whether the integer was prime.

Answer:

A prime number is a positive integer that has two divisors only, the number 1 and the prime number itself. The basic test for a prime number is to divide a candidate number by all the integers from 2 up to the square root of the candidate number. If any divisions give a remainder of 0 then the number is not prime.

Several small optimisations can be made. Firstly any even number is immediately rejected by checking to see if dividing by two leaves a remainder of zero. Second, the search performed by the while loop only tries to divide using odd numbers, as any multiple of an even number must be an even number and hence not prime.

```
public class Q17
{
  public boolean isPrime(long n)
  {
    if (n < 2L)
    {
      return false;
    }
    if (n == 2L)
    {
      return true;
    }
    if ((n % 2L) == 0)
    {
      return false;
    }
    long divisor = 3L;
    final long limit = new Double(Math.sqrt(n)).longValue();
    while (divisor <= limit)
    {
      if ((n % divisor) == 0)
      {
        return false;
      }
      divisor += 2L;
    }
    return true;
  }

  public void checkPrime()
  {
    Input input = new Input();
    System.out.print("Enter a number: ");
```

```
    long candidate = input.nextLong();
    if (isPrime(candidate))
    {
      System.out.println(candidate + " is a prime number!");
    }
    else
    {
      System.out.println(candidate + " is not a prime number.");
    }
  }

  public static void main(final String[] args)
  {
    new Q17().checkPrime();
  }
}
```

The method above performs a brute force test. There are a number of much more efficient ways of testing if a number is prime but they require the use of more sophisticated data structures.

b) Next, using your prime method, write a program that finds all the prime numbers that can be represented by an integer of type long.

Notes: This is not quite as easy as it might appear, especially if you want the program to produce results quickly. Search the web for information about prime numbers and algorithms for finding them - there are some excellent web sites.

Answer:

Here is a program that simply tests all number from three upwards. It will take a long time to run!

```
public class Q17a
{
  public boolean isPrime(long n)
  {
    if (n < 2L)
    {
      return false;
    }
    if (n == 2L)
    {
      return true;
    }
    if ((n % 2L) == 0)
    {
      return false;
    }
    long divisor = 3L;
    long limit = new Double(Math.sqrt(n)).longValue();
    while (divisor <= limit)
    {
      if ((n % divisor) == 0)
      {
        return false;
      }
      divisor += 2L;
```

```
    }
    return true;
  }

  public void findPrimes()
  {
    long count = 0L;
    for (long candidate = 3L; candidate < Long.MAX_VALUE; candidate++)
    {
      if (isPrime(candidate))
      {
        System.out.println(candidate + " is Prime");
      }
    }
  }

  public static void main(final String[] args)
  {
    new Q17a().findPrimes();
  }
}
```

The constant values are written as ‘2L’ or ‘3L’ in order to force the use of type long. The ‘L’ is used to specify the type is long and not the default of int.

Q1.18 Write a program that reads an integer between 0 and 999 and "verbalises it". For example, if the program is given 123 it would display "one hundred and twenty three".

Hint: Write methods to deal with ranges of numbers, such as single digits between "zero" and "nine", numbers between 10 and 19 and so on.

Answer:

This is a well-known programming exercise. Here is one solution:

```
import java.util.HashMap;
import java.util.Map;

public class Q18
{
  private static Map<Integer,String> numbers =
    new HashMap<Integer,String>();

  static {
    numbers.put(0,"zero");
    numbers.put(1,"one");
    numbers.put(2,"two");
    numbers.put(3,"three");
    numbers.put(4,"four");
    numbers.put(5,"five");
    numbers.put(6,"six");
    numbers.put(7,"seven");
    numbers.put(8,"eight");
    numbers.put(9,"nine");
    numbers.put(10,"ten");
    numbers.put(11,"eleven");
```

```
  numbers.put(12,"twelve");
  numbers.put(13,"thirteen");
  numbers.put(14,"fourteen");
  numbers.put(15,"fifteen");
  numbers.put(16,"sixteen");
  numbers.put(17,"seventeen");
  numbers.put(18,"eighteen");
  numbers.put(19,"nineteen");
  numbers.put(20,"twenty");
  numbers.put(30,"thirty");
  numbers.put(40,"forty");
  numbers.put(50,"fifty");
  numbers.put(60,"sixty");
  numbers.put(70,"seventy");
  numbers.put(80,"eighty");
  numbers.put(90,"ninety");
}

private String verbaliseLessThanTwenty(int n)
{
  return numbers.get(n);
}

private String verbaliseTwentyToNinetyNine(int n)
{
  String result = numbers.get(n / 10 * 10);
  if ((n % 10) != 0)
    { result += " " + verbaliseLessThanTwenty(n % 10); }
  return result;
}

private String verbaliseOneHundredToNineHundredAndNinetyNine(int n)
{
  String result = verbaliseLessThanTwenty(n / 100) + " hundred";
  if ((n % 100) == 0) { return result; }
  result += " and ";
  if ((n % 100) < 20)
    { return result + verbaliseLessThanTwenty(n % 100); }
  return result + verbaliseTwentyToNinetyNine(n % 100);
}

public String verbalise(String number) throws IllegalArgumentException
{
  int n = 0;
  try
  {
    n = Integer.parseInt(number);
  }
  catch(NumberFormatException e)
  {
    throw new IllegalArgumentException("Not a number");
  }
  if ((n < 0) || (n > 999))
    { throw new IllegalArgumentException("Out of range"); }
  if (n < 21) { return verbaliseLessThanTwenty(n); }
  if (n < 100) { return verbaliseTwentyToNinetyNine(n); }
```

```
    return verbaliseOneHundredToNineHundredAndNinetyNine(n);
  }

  public void go()
  {
    Input input = new Input();
    System.out.print("Enter number: ");
    String number = input.nextLine();
    try
    {
      System.out.println(verbalise(number));
    }
    catch (IllegalArgumentException e)
    {
      System.out.println
        ("Cannot verbalise the input: " + e.getMessage());
    }
  }

  public static void main(String[] args)
  {
    new Q18().go();
  }
}
```