Homework 2. Exercises on Java Basics

Writing Good Programs

The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.
It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these good programs. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.
Pay particular attention to:

1. **Coding style**:
   - Read Java code convention: "Java Style and Commenting Guide".
   - Follow the Java Naming Conventions for variables, methods, and classes STRICTLY. Use CamelCase for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., radius) and class names (e.g., Circle). Use verbs for methods (e.g., getArea(), isEmpty()).
   - **Use Meaningful Names**: Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like i, j, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and i for array index. Use meaningful names like row and col (instead of x and y, i and j, x1 and x2), numStudents (not n), maxGrade, size (not n), and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use books for an array of books, and book for each item).
   - Use consistent indentation and coding style. Many IDEs (such as Eclipse / NetBeans) can re-format your source codes with a single click.
2. **Program Documentation**: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.
3. The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)).

1 More Exercises

1.1 Matrices (2D Arrays)

Similar to *Math* class, write a Matrix library that supports matrix operations (such as addition, subtraction, multiplication) via 2D arrays. The operations shall support both *double* and *int*. Also write a test class to exercise all the operations programmed.

Hints

```
public class Matrix {
  // Method signatures
  public static void print(int[][] matrix);
  public static void print(double[][] matrix);

  // Used in add(), subtract()
  public static boolean haveSameDimension(int[][] matrix1,
                                          int[][] matrix2);
  public static boolean haveSameDimension(double[][] matrix1,
                                          double[][] matrix2);

  public static int[][] add(int[][] matrix1, int[][] matrix2);
  public static double[][] add(double[][] matrix1,
                               double[][] matrix2);

  public static int[][] subtract(int[][] matrix1, int[][] matrix2);
  public static double[][] subtract(double[][] matrix1,
                                    double[][] matrix2);

  public static int[][] multiply(int[][] matrix1, int[][] matrix2);
  public static double[][] multiply(double[][] matrix1,
                                    double[][] matrix2);
  ......
}
```

1.2 Trigonometric Series

Write a method to compute $\sin(x)$ and $\cos(x)$ using the following series expansion, in a class called **TrigonometricSeries**. The signatures of the methods are:

```
// x in radians, NOT degrees
public static double sin(double x, int numTerms);
public static double cos(double x, int numTerms);
```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$
$$\cos(x) = x - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Compare the values computed using the series with the JDK methods *Math.sin()*, *Math.cos()* at $x = 0, \pi/6, \pi/4, \pi/3, \pi/2$ using various numbers of terms.

Hints
Do not use int to compute the factorial; as factorial of 13 is outside the int range. Avoid generating large numerator and denominator. Use double to compute the terms as:

$$\frac{x^n}{n!} = \left(\frac{x}{n}\right)\left(\frac{x}{n-1}\right)\ldots\left(\frac{x}{1}\right).$$

1.3 Exponential Series

Write a method to compute the sum of the series in a class called **SpecialSeries**. The signature of the method is:

```java
public static double specialSeries(double x, int numTerms);
```

$$x + \frac{1}{2} \times \frac{x^3}{3} + \frac{1 \times 3}{2 \times 4} \times \frac{x^5}{5} + \frac{1 \times 3 \times 5}{2 \times 4 \times 6} \times \frac{x^7}{7} + \frac{1 \times 3 \times 5 \times 7}{2 \times 4 \times 6 \times 8} \times \frac{x^9}{9} + \cdots ; -1 \leq x \leq 1.$$

1.4 FactorialInt (Handling Overflow)

Write a program called **FactorialInt** to list all the factorials that can be expressed as an *int* (i.e., 32-bit signed integer in the range of $[-2147483648,\ 2147483647]$). Your output shall look like:

Command window

```
The factorial of 1 is 1
The factorial of 2 is 2
...
The factorial of 12 is 479001600
The factorial of 13 is out of range
```

Hints
The maximum and minimum values of a 32-bit int are kept in constants *Integer.MAX_VALUE* and *Integer.MIN_VALUE*, respectively. Try these statements:

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use $F(n) * (n + 1) > Integer.MAX\_VALUE$ to check for overflow. Instead, overflow occurs for $F(n+1)$ if ($Integer.MAX\_VALUE$ / Factorial(n)) $< (n + 1)$, i.e., no more room for the next number.

Try
Modify your program called **FactorialLong** to list all the factorial that can be expressed as a long (64-bit signed integer). The maximum value for long is kept in a constant called *Long.MAX_VALUE*.

1.5 FibonacciInt (Handling Overflow)

Write a program called **FibonacciInt** to list all the Fibonacci numbers, which can be expressed as an int (i.e., 32-bit signed integer in the range of $[-2147483648, 2147483647]$). The output shall look like:

Command window

```
F(0) = 1
F(1) = 1
F(2) = 2
...
F(45) = 1836311903
F(46) is out of the range of int
```

Hints
The maximum and minimum values of a 32-bit int are kept in constants *Integer.MAX_VALUE* and *Integer.MIN_VALUE*, respectively. Try these statements:

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use $F(n) = F(n-1) + F(n-2) > Integer.MAX\_VALUE$ to check for overflow. Instead, overflow occurs for $F(n)$ if *Inte-*

ger.MAX_VALUE $-F(n-1) < F(n-2)$ (i.e., no more room for the next Fibonacci number).

Try
Write a similar program called **TribonacciInt** for Tribonacci numbers.

1.6 Number System Conversion

Write a method call *toRadix()* which converts a positive integer from one radix into another. The method has the following header:

```java
// The input and output are treated as String.
public static String toRadix(String in, int inRadix, int outRadix)
```

Write a program called **NumberConversion**, which prompts the user for an input string, an input radix, and an output radix, and display the converted number. The output shall look like:

Command window

```
Enter a number and radix: A1B2
Enter the input radix: 16
Enter the output radix: 2
"A1B2" in radix 16 is "1010000110110010" in radix 2.
```

1.7 NumberGuess

Write a program called **NumberGuess** to play the number guessing game. The program shall generate a random number between 0 and 99. The player inputs his/her guess, and the program shall response with "Try higher", "Try lower" or "You got it in n trials" accordingly. For example:

Command window

```
java NumberGuess
Key in your guess:
50
Try higher
70
Try lower
65
Try lower
61
You got it in 4 trials!
```

Hints

Use *Math.random()* to produce a random number in double between 0.0 (inclusive) and 1.0 (exclusive). To produce an int between 0 and 99, use:

```java
final int SECRET_NUMBER = (int)(Math.random()*100);  // truncate to int
```

1.8 WordGuess

Write a program called **WordGuess** to guess a word by trying to guess the individual characters. The word to be guessed shall be provided using the command-line argument. Your program shall look like:

Command window

```
java WordGuess testing
Key in one character or your guess word: t
Trial 1: t__t___
Key in one character or your guess word: g
Trial 2: t__t__g
Key in one character or your guess word: e
Trial 3: te_t__g
Key in one character or your guess word: testing
Congratulation!
You got in 4 trials
```

Hints

1. Set up a *boolean* array (of the length of the word to be guessed) to indicate the positions of the word that have been guessed correctly.
2. Check the length of the input *String* to determine whether the player enters a single character or a guessed word. If the player enters a single character, check it against the word to be guessed, and update the boolean array that keeping the result so far.

Try

Try retrieving the word to be guessed from a text file (or a dictionary) randomly.

1.9 DateUtil

Complete the following methods in a class called **DateUtil**:

- *boolean isLeapYear(int year)*: returns *true* if the given year is a leap year. A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

- *boolean isValidDate(int year, int month, int day)*: returns true if the given year, month and day constitute a given date. Assume that year is between 1 and 9999, month is between 1 (Jan) to 12 (Dec) and day shall be between 1 and 28|29|30|31 depending on the month and whether it is a leap year.
- *int getDayOfWeek(int year, int month, int day)*: returns the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT, for the given date. Assume that the date is valid.
- *String toString(int year, int month, int day)*: prints the given date in the format "xxxday d mmm yyyy", e.g., "Tuesday 14 Feb 2012". Assume that the given date is valid.

Hints

To find the day of the week (Reference: Wiki "Determination of the day of the week"):

1. Based on the first two digit of the year, get the number from the following "century" table.

1700-	1800-	1900-	2000-	2100-	2200-	2300-	2400-
4	2	0	6	4	2	0	6

   Take note that the entries $4, 2, 0, 6$ repeat.

2. Add to the last two digit of the year.
3. Add to "the last two digit of the year divide by 4, truncate the fractional part".
4. Add to the number obtained from the following month table:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Non-Leap Year	0	3	3	6	1	4	6	2	5	0	3	5
Leap Year	6	2	3	6	1	4	6	2	5	0	3	5

5. Add to the day.
6. The sum modulus 7 gives the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT.

For example: 2012, Feb, 17

EX

```
(6 + 12 + 12/4 + 2 + 17) % 7 = 5 (Fri)
```

The skeleton of the program is as follows:

```java
/* Utilities for Date Manipulation */
public class DateUtil {

  // Month's name - for printing
  public static String[] strMonths
    = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
       "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

  // Number of days in each month (for non-leap years)
  public static int[] daysInMonths
    = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

  // Returns true if the given year is a leap year
  public static boolean isLeapYear(int year) { ...... }

  // Return true if the given year, month, day is a valid date
  // year: 1-9999
  // month: 1(Jan)-12(Dec)
  // day: 1-28|29|30|31. The last day depends on year and month
  public static boolean isValidDate(int year, int month, int day) { ...... }

  // Return the day of the week, 0:Sun, 1:Mon, ..., 6:Sat
  public static int getDayOfWeek(int year, int month, int day) { ...... }

  // Return String "xxxday d mmm yyyy" (e.g., Wednesday 29 Feb 2012)
  public static String printDate(int year, int month, int day) { ...... }

  // Test Driver
  public static void main(String[] args) {
    System.out.println(isLeapYear(1900)); // false
    System.out.println(isLeapYear(2000)); // true
    System.out.println(isLeapYear(2011)); // false
    System.out.println(isLeapYear(2012)); // true

    System.out.println(isValidDate(2012, 2, 29)); // true
    System.out.println(isValidDate(2011, 2, 29)); // false
    System.out.println(isValidDate(2099, 12, 31)); // true
    System.out.println(isValidDate(2099, 12, 32)); // false

    System.out.println(getDayOfWeek(1982, 4, 24)); // 6:Sat
    System.out.println(getDayOfWeek(2000, 1, 1)); // 6:Sat
    System.out.println(getDayOfWeek(2054, 6, 19)); // 5:Fri
    System.out.println(getDayOfWeek(2012, 2, 17)); // 5:Fri

    System.out.println(toString(2012, 2, 14)); // Tuesday 14 Feb 2012
  }
}
```

Notes
You can compare the day obtained with the Java's Calendar class as follows:

```java
// Construct a Calendar instance with the given year, month and day
Calendar cal = new GregorianCalendar(year, month - 1, day);  // month
    ↪ is 0-based
// Get the day of the week number: 1 (Sunday) to 7 (Saturday)
int dayNumber = cal.get(Calendar.DAY_OF_WEEK);
String[] calendarDays = { "Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday" };
// Print result
System.out.println("It is " + calendarDays[dayNumber - 1]);
```

The calendar we used today is known as Gregorian calendar, which came into effect in October 15, 1582 in some countries and later in other countries. It replaces the Julian calendar. 10 days were removed from the calendar, i.e., October 4, 1582 (Julian) was followed by October 15, 1582 (Gregorian). The only difference between the Gregorian and the Julian calendar is the "leap-year rule". In Julian calendar, every four years is a leap year. In Gregorian calendar, a leap year is a year that is divisible by 4 but not divisible by 100, or it is divisible by 400, i.e., the Gregorian calendar omits century years which are not divisible by 400. Furthermore, Julian calendar considers the first day of the year as march 25th, instead of January 1st.
This above algorithm work for Gregorian dates only. It is difficult to modify the above algorithm to handle pre-Gregorian dates. A better algorithm is to find the number of days from a known date.

2 Exercises on Recursion

n programming, a recursive function (or method) calls itself. The classical example is factorial(n), which can be defined recursively as $f(n) = n * f(n-1)$. Nonetheless, it is important to take note that a recursive function should have a terminating condition (or base case), in the case of factorial, $f(0) = 1$. Hence, the full definition is:

```
factorial(n) = 1, for n = 0
factorial(n) = n * factorial(n - 1), for all n > 0
```

For example, suppose $n = 5$:

```
// Recursive call
factorial(5) = 5 * factorial(4)
factorial(4) = 4 * factorial(3)
factorial(3) = 3 * factorial(2)
factorial(2) = 2 * factorial(1)
factorial(1) = 1 * factorial(0)
factorial(0) = 1  // Base case

\noindent // Unwinding
factorial(1) = 1 * 1 = 1
factorial(2) = 2 * 1 = 2
factorial(3) = 3 * 2 = 6
factorial(4) = 4 * 6 = 24
factorial(5) = 5 * 24 = 120 (DONE)
```

2.1 Factorial Recusive

Write a recursive method called *factorial()* to compute the factorial of the given integer.

```java
public static int factorial(int n)
```

The recursive algorithm is:

```
factorial(n) = 1, for n = 0
factorial(n) = n * factorial(n-1), for all n > 0
```

Compare your code with the iterative version of the factorial():

```
factorial(n) = 1 * 2 * 3 * \dots * n
```

Hints
Writing recursive function is straight forward. You simply translate the recursive definition into code with return.

```java
// Return the factorial of the given integer, recursively
public static int factorial(int n) {
```

```java
    if (n == 0) {
      return 1;    // base case
    } else {
      return n * factorial(n-1);  // call itself
    }
    // or one liner
    // return (n == 0) ? 1 : n*factorial(n-1);
  }
```

or

```java
  // Return the factorial of the given integer, recursively
  public static int factorial(int n) {
    if (n == 0) {
      return 1;    // base case
    }

    return n * factorial(n-1);  // call itself

    // or one liner
    // return (n == 0) ? 1 : n*factorial(n-1);
  }
```

Notes

1. Recursive version is often much shorter.
2. The recursive version uses much more computational and storage resources, and it need to save its current states before each successive recursive call, so as to unwind later.

2.2 Fibonacci (Recursive)

Write a recursive method to compute the Fibonacci number of n, defined as follows:

```
F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2) for n >= 2
```

Compare the recursive version with the iterative version written earlier.

Hints

```java
// Translate the recursive definition into code with return statements
public static int fibonacci(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return fibonacci(n-1) + fibonacci(n-2);
  }
}
```

or

```java
// Translate the recursive definition into code with return statements
public static int fibonacci(int n) {
  if (n == 0) {
    return 0;
  }

  if (n == 1) {
    return 1;
  }

  return fibonacci(n-1) + fibonacci(n-2);
}
```

2.3 Length of a Running Number Sequence (Recursive)

A special number sequence is defined as follows:

```
S(1) = 1
S(2) = 12
S(3) = 123
S(4) = 1234
......
S(9) = 123456789           // length is 9
S(10) = 12345678910        // length is 11
S(11) = 1234567891011      // length is 13
S(12) = 123456789101112    // length is 15
......
```

Write a recursive method to compute the length of $S(n)$, defined as follows:

```
len(1) = 1
len(n) = len(n-1) + numOfDigits(n)
```

2.4 GCD (Recursive)

Write a recursive method called *gcd()* to compute the greatest common divisor of two given integers.

```java
public static void int gcd(int a, int b)
```

```
gcd(a, b) = a, if b = 0
gcd(a, b) = gcd(b, remainder(a,b)), if b > 0
```

3 Exercises on Algorithms - Sorting and Searching

fficient sorting and searching are big topics, typically covered in a course called "Data Structures and Algorithms". There are many searching and sorting algorithms available, with their respective strengths and weaknesses. See Wikipedia "Sorting Algorithms" and "Searching Algorithms" for the algorithms, examples and illustrations.
JDK provides searching and sorting utilities in the Arrays class (in package java.util), such as Arrays.sort() and Arrays.binarySearch() - you don't have to write your searching and sorting in your production program. These exercises are for academic purpose and for you to gain some understandings and practices on these algorithms.

3.1 Linear Search

Write the following linear search methods to search for a key value in an array, by comparing each item with the search key in the linear manner. Linear search is applicable to unsorted list. (Reference: Wikipedia "Linear Search".)

```java
// Return true if the key is found inside the array
public static boolean linearSearch(int[] array, int key)

// Return the array index, if key is found; or 0 otherwise
public static int linearSearchIndex(int[] array, int key)
```

Also write a test driver to test the methods.

3.2 Recursive Binary Search

(Reference: Wikipedia "Binary Search") Binary search is only applicable to a sorted list. For example, suppose that we want to search for the item 18 in the list {1114161820252830344045}:

Create two indexes: firstIdx and lastIdx , initially pointing at the first and last elements
{11 14 16 18 20 25 28 30 34 40 45}
F M L
Compute middleIdx = (firstIdx + lastIdx) / 2
Compare the key (K) with the middle element (M)
If K = M, return true
else if K < M, set firstIdx = middleIndex
else if K > M, set firstIdx = middleIndex
{11 14 16 18 20 25 28 30 34 40 45}
F M L
Recursively repeat the search between the new firstIndex and lastIndex.
Terminate with not found when firstIndex = lastIndex.
{11 14 16 18 20 25 28 30 34 40 45}
F M L

Write a recursive function called *binarySearch()* as follows:

```java
// Return true if key is found in the array in the range of fromIdx (
    ↪ inclusive) to toIdx (exclusive)
public boolean binarySearch(int[] array, int key, int fromIdx, int
    ↪ toIdx)
```

Use the following pseudocode implementation:

```
If fromIdx = toIdx - 1    // Terminating one-element list
  if key = array[fromIdx], return true
  else, return false (not found)
else
  middleIdx = (fromIdx + toIdx) / 2
  if key = array[middleIdx], return true
  else if key < array[middleIdx], toIdx = middleIdx
  else firstIdx = middleIdx + 1
  binarySearch(array, key, fromIdx, toIdx)  // recursive call
```

Also write an overloaded method which uses the above to search the entire array:

```java
// Return true if key is found in the array
public boolean binarySearch(int[] array, int key)
```

Write a test driver to test the methods.

3.3 Bubble Sort

(Reference: Wikipedia "Bubble Sort") The principle of bubble sort is to scan the elements from left-to-right, and whenever two adjacent elements are out-of-order, they are swapped. Repeat the passes until no swap are needed. For example, given the list {9 2 4 1 5}, to sort in ascending order:

Pass 1:
9 2 4 1 5 → 2 9 4 1 5
2 9 4 1 5 → 2 4 9 1 5
2 4 9 1 5 → 2 4 1 9 5
2 4 1 9 5 → 2 4 1 5 9 (After Pass 1, the largest item sorted on the right - bubble to the right)
Pass 2:
2 4 1 5 9 → 2 4 1 5 9
2 4 1 5 9 → 2 1 4 5 9
2 1 4 5 9 → 2 1 4 5 9
2 1 4 5 9 → 2 1 4 5 9 (After Pass 2, the 2 largest items sorted on the right)
Pass 3:
2 1 4 5 9 → 1 2 4 5 9
1 2 4 5 9 → 1 2 4 5 9
1 2 4 5 9 → 1 2 4 5 9
1 2 4 5 9 → 1 2 4 5 9 (After Pass 3, the 3 largest items sorted on the right)
Pass 4:
1 2 4 5 9 → 1 2 4 5 9
1 2 4 5 9 → 1 2 4 5 9
1 2 4 5 9 → 1 2 4 5 9
1 2 4 5 9 → 1 2 4 5 9 (After Pass 4, the 4 largest items sorted on the right)
No Swap in Pass 4. Done.

See Wikipedia "Bubble Sort" for more examples and illustration.
Write a method to sort an int array (in place) with the following signature:

```java
public static void bubbleSort(int[] array)
```

Use the following pesudocode implementation:

```
function bubbleSort(array)
n = length(array)
boolean swapped    // boolean flag to indicate swapping occurred during
    ↪ a pass
do {
  swapped = false  // reset for each pass
  for (i = 1; i < n; ++i) {
    // Swap if this pair is out of order
    if array[i-1] > array[i] {
      swap( A[i-1], A[i] )
      swapped = true   // update flag
    }
  }
  n = n - 1  // One item sorted after each pass
} while (swapped)  // repeat another pass if swapping occurred,
    ↪ otherwise done
```

3.4 Selection Sort

(Reference: Wikipedia "Selection Sort") This algorithm divides the lists into two parts: the left-sublist of items already sorted, and the right-sublist for the remaining items. Initially, the left-sorted-sublist is empty, while the right-unsorted-sublist is the entire list. The algorithm proceeds by finding the smallest (or largest) items from the right-unsorted-sublist, swapping it with the leftmost element of the right-unsorted-sublist, and increase the left-sorted-sublist by one.
For example, given the list {9 6 4 1 5}, to sort in ascending order:

9 6 4 1 5 → 1 6 4 9 5
1 6 4 9 5 → 1 4 6 9 5
1 4 6 9 5 → 1 4 5 9 6
1 4 5 9 6 → 1 4 5 6 9
1 4 5 6 9 → DONE
1 4 5 6 9

Write a method to sort an int array (in place) with the following signature:

```java
public static void selectionSort(int[] array)
```

3.5 Insertion Sort

(Reference: Wikipedia "Insertion Sort") Similar to the selection sort, but extract the leftmost element from the right-unsorted-sublist, and insert into the correct location of the left-sorted-sublist.
For example, given {9 6 4 1 5 2 7}, to sort in ascending order:

```
{} {9 6 4 1 5 2 7} → {9} {6 4 1 5 2 7}
{9} {6 4 1 5 2 7}  → {6 9} {4 1 5 2 7}
{6 9} {4 1 5 2 7}  → {4 6 9} {1 5 2 7}
{4 6 9} {1 5 2 7}  → {1 4 6 9} {5 2 7}
{1 4 6 9} {5 2 7}  → {1 4 5 6 9} {2 7}
{1 4 5 6 9} {2 7}  → {1 2 4 5 6 9} {7}
{1 2 4 5 6 9} {7}  → {1 2 4 5 6 7 9} {}
{1 2 4 5 6 7 9} {} → Done
```

Write a method to sort an int array (in place) with the following signature:

```java
public static void insertionSort(int[] array)
```

4 Exercises on Algorithms - Number Theory

4.1 Perfect and Deficient Numbers

A positive integer is called a perfect number if the sum of all its factors (excluding the number itself, i.e., proper divisor) is equal to its value. For example, the number 6 is perfect because its proper divisors are 1, 2, and 3, and $6 = 1 + 2 + 3$; but the number 10 is not perfect because its proper divisors are 1, 2, and 5, and $10 \neq 1 + 2 + 5$.
A positive integer is called a deficient number if the sum of all its proper divisors is less than its value. For example, 10 is a deficient number because $1 + 2 + 5 < 10$; while 12 is not because $1 + 2 + 3 + 4 + 6 > 12$.
Write a boolean method called *isPerfect(int aPosInt)* that takes a positive integer, and return true if the number is perfect. Similarly, write a boolean method called *isDeficient(int aPosInt)* to check for deficient numbers.

```java
public static boolean isPerfect(int aPosInt);
public static boolean isDeficient(int aPosInt);
```

Using the methods, write a program called **PerfectNumberList** that prompts user for an upper bound (a positive integer), and lists all the perfect numbers less than or equal to this upper bound. It shall also list all the numbers that are neither deficient nor perfect. The output shall look like:

Command window

```
Enter the upper bound: 1000
These numbers are perfect:
6 28 496
[3 perfect numbers found (0.30%)]

These numbers are neither deficient nor perfect:
12 18 20 24 30 36 40 42 48 54 56 60 66 70 72 78 80 ......
[246 numbers found (24.60%)]
```

4.2 Prime Numbers

A positive integer is a prime if it is divisible by 1 and itself only. Write a boolean method called *isPrime(int aPosInt)* that takes a positive integer and returns *true* if the number is a prime. Write a program called **PrimeList** that prompts the user for an upper bound (a positive integer), and lists all the primes less than or equal to it. Also display the percentage of prime (rounded to 2 decimal places). The output shall look like:

Command window

```
Please enter the upper bound: 10000
1
2
3
......
......
9967
9973
[1230 primes found (12.30%)]
```

Hints

To check if a number n is a prime, the simplest way is try dividing n by 2 to sqrt(n).

4.3 Prime Factors

Write a boolean method called *isProductOfPrimeFactors(int aPosInt)* that takes a positive integer, and return *true* if the product of all its prime factors (excluding 1 and the number itself) is equal to its value. For example, the method returns true for 30 ($30 = 2 \times 3 \times 5$) and false for 20 ($20 \neq 2 \times 5$). You may need to use the *isPrime()* method in the previous exercise.

Write a program called **PerfectPrimeFactorList** that prompts user for an upper bound. The program shall display all the numbers (less than or equal to the upper bound) that meets the above criteria. The output shall look like:

Command window

```
Enter the upper bound: 100
These numbers are equal to the product of prime factors:
1 6 10 14 15 21 22 26 30 33 34 35 38 39 42 46 51 55 57 58 62 65 66 69 70 74 77 78 82 85 86 87 91 93 94 95
[36 numbers found (36.00%)]
```

4.4 Greatest Common Divisor (GCD)

One of the earlier known algorithms is the Euclid algorithm to find the GCD of two integers (developed by the Greek Mathematician Euclid around 300BC). By definition, GCD(a, b) is the greatest factor that divides both a and b. Assume that a and b are positive integers, and $a \geq b$, the Euclid algorithm is based on these two properties:

```
GCD(a, 0) = a
GCD(a, b) = GCD(b, a mod b), where (a mod b) denotes the remainder of a divides by b.
```

For example,

```
GCD(15, 5) = GCD(5, 0) = 5
GCD(99, 88) = GCD(88, 11) = GCD(11, 0) = 11
GCD(3456, 1233) = GCD(1233, 990) = GCD(990, 243) = GCD(243, 18) = GCD(18, 9) = GCD(9, 0) = 9
```

The pseudocode for the Euclid algorithm is as follows:

```
GCD(a, b)    // assume that a >= b
while (b != 0) {
  // Change the value of a and b: a <- b, b <- a mod b, and repeat until b is 0
  temp <- b
  b <- a mod b
  a <- temp
}
// after the loop completes, i.e., b is 0, we have GCD(a, 0)
GCD is a
```

Write a method called *gcd()* with the following signature:

```java
public static int gcd(int a, int b)
```

Your methods shall handle arbitrary values of a and b, and check for validity.