1. Homework 1

Posted: June/3/2020
Due: August/25/2020 24.00

The solutions for this homework are due August/25/2020 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Submission of your Homework

This course uses the myCourses submission functionality to allow students to electronically submit their work.

I suggest to submit often, meaning submit long before the deadline. We will only accept submitted code. You will receive 0 points for the hw, if your code is not submitted.

1.2. Programming Standard

Please make sure that your code is compatible with the Coding Standard (https://www.cs.rit.edu/~f2y-grd/java-coding-standard.html)

We will take points of, if your code does not follow this standard.

1.3. Teamwork

All work has to be submitted as a team of 2. You have to appear to the grading sessions on time. You have to select a grading slot during the first week. A schedule will be posted at the grad lab door at the beginning of the first week.

You will receive 0 points if you are late for your grading session.

The graders determine who answers the questions.

The signup sheet for your team can be found here: https://docs.google.com/spreadsheets/d/1So-jVaBhfyG_IiGCWTDM4wgazpbG_nTPVQGk61rdlf0/edit?usp=sharing (https://docs.google.com/spreadsheets/d/1So-jVaBhfyG_IiGCWTDM4wgazpbG_nTPVQGk61rdlf0/edit?usp=sharing)

1.4. Homework 1.1 (10 Points)

Objective: Compilation of a Java program, designing, implementing, and testing of an algorithm.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

All homework are submitted as a team of 2.

Definition: Prime Number

A number p is a prime number if p has the following properties:

• p must be a integer

• $p > 1$ and

• the factors of p are 1 and itself.

The following program prints out all prime numbers in the range of [2 ... 10]:

```
 1      class Prime {
 2
 3        public static boolean isPrime(int n) {
 4
 5              for ( int index = 2; index < n; index ++ ) {
 6                      if ( n % index  == 0 )
 7                              return false;
 8              }
 9
10              return true;
11        }
12        public static void main( String args[] ) {
13          for ( int index = 2; index <= 10; index ++ )
14              if ( isPrime(index) )
15                      System.out.println(index + " " );
16        }
17      }
```

Source Code: Src/21/Prime.java

Your Work:

Modify *Prime.java* in such a way that is tests all numbers between *n* 3 and 73939233 for the following properties:

• *n* is a prime number

• Remove the right digit and and this number is also a prime number. Continue this process until there is only one digit left or the number is not a prime number.

• Your program has to print all numbers which have these properties.

Explanation:

An example for a number which has these properties is 233.

• 233 is a prime number

• 23 is a prime number

• 2 is a prime number

An example for a number which does not these properties is 397.

• 397 is a prime number

• 39 is not a prime number (3 * 13)

Example:

An example of a solution execution:

```
% java Prime
3 has the properties.
5 has the properties.
7 has the properties.
23 has the properties.
29 has the properties.
31 has the properties.
37 has the properties.
53 has the properties.
59 has the properties.
```

```
71 has the properties.
73 has the properties.
79 has the properties.
233 has the properties.
239 has the properties.
293 has the properties.
311 has the properties.
313 has the properties.
317 has the properties.
373 has the properties.
379 has the properties.
 ...
```

Idea for a Solution:

A statement like *thisNumber = (int)(thisNumber / 10)* cuts of the last digit.

Requirements:

• You have to name your program Prime.java

• You can only use basic arithmetic operations, casting, boolean expressions, print statements, Math.sqrt, basic types and native arrays.

• You can not use any publicly available class or library which can determine if a number is a prime number.

• Your program has to compute the properties of the number. In other words your program can not be something like:

```
 1      class PrimeWrong {
 2
 3        public static void main( String args[] ) {
 4              System.out.println("3 has the properties.");
 5              System.out.println("5 has the properties.");
 6              System.out.println("7 has the properties.");
 7              System.out.println("23 has the properties.");
 8              System.out.println("29 has the properties.");
 9              System.out.println("...");
10        }
11      }
```

Source Code: Src/21/PrimeWrong.java

Submission:

Submit your files via myCourses.

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
1       public class Prime {
2
3           // final static int MAXIMUM = 73939133 + 1;
4           final static int MAXIMUM = 400;
5           final static int MINIMUM = 1;
6           final static boolean  thisNumberIsAprimeNumber[] = new boolean[ MAXIMUM];
7
8           private static boolean isItAPrimeNumber(int n)      {       // is not used
9               boolean rValue = true;
10              int index = 1;
11              if ( n <= 1 )
12                      return false;
13              if ( n == 2 )
14                      rValue = true;
15              while ( ( ++index <= ( (int)Math.sqrt(n) ) ) &&  rValue ) {
16                      if ( n % index == 0 )   {
17                          rValue = false;
18                      }
19              }
20              return rValue;
21          }
22          private static void initPrimeOrNotSieve()   {
23              for ( int index = 0; index < thisNumberIsAprimeNumber.length; index ++ )
24                      thisNumberIsAprimeNumber[index] = false;
25
26              for ( int factor1 = 2; factor1 <=  ( (int)Math.sqrt(thisNumberIsAprimeNumb
27                      for ( int factor2 = 2; factor2 * factor1 <   thisNumberIsAprimeNum
28                              thisNumberIsAprimeNumber[factor1 * factor2 ] = true;
29                      }
30              }
31          }
32          private static boolean hasProperty(int thisNumber) {
33
34              while ( ( thisNumber > 1 ) && ! thisNumberIsAprimeNumber[thisNumber] )  {
35                      thisNumber = thisNumber / 10;
36              }
37              return ( thisNumber == 0 );
38          }
39          private static void findPrimesWithTheseProperties() {
40              initPrimeOrNotSieve();
41              // print();
42              for ( int index = MINIMUM; index < MAXIMUM; index +=2 ) {
43                      if ( ! thisNumberIsAprimeNumber[index] && hasProperty(index)    )
44                              System.out.println(index + " has the properties.");
45              }
46          }
47          private static void print() {
48              for ( int index = 0; index < thisNumberIsAprimeNumber.length; index ++ )
49                      System.out.println(index + ":   " + thisNumberIsAprimeNumber[index
50              }
51          }
52
53          public static void main( String[] args ) {
54              new Prime().findPrimesWithTheseProperties();
```

```
55            }
56        }
57
58
```

Source Code: Src/21/PrimeSol.java

1.5. Homework 1.2 (10 Points)

Objective: Creating and implementing an algorithm.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

This homework was inspired by having too many coins in my wallet. You have a given, fixed amount of coins and you can only pay the cashier with these coins, and the cashier accepts only the exact amount. If can have the exact amount you have to use the maximum number of coins possible to pay the bill.

The coins distribution will be initialized to the original state after each paying process.

Explanation:

Let's assume you have the following coins: { 1, 1, 2, 5, 25, 25, 25 } cents, and you have to pay 7 cents.

1 + 1 + 5 == 7, and

2 + 5 == 7.

The sequence { 1, 1, 5 } contains 3 coins and the sequence { 2, 5 } contains 2 coins. You have to use the sequence { 1, 1, 5 } to pay.

Let's assume you have the following coins: { 1, 1, 2, 5, 25, 25, 25 } cents, and you have to pay 11 cents. This is not possible.

Your Work:

Your work is to write a program which determines if a sequence of coins exist adding up to a given value. Your program has to print sequence, or the longest sequence if more than one sequence exist.

Assume you have the following coins: *coins = { 1, 1, 2, 5, 25, 25, 25 };* and you have to pay the following sums *toPay = { 0, 1, 4, 5, 7, 8 };*.

```
0 cents: can not be paid
1 cents: = 1 cents
4 cents: = 2 cents 1 cents 1 cents
5 cents: = 5 cents
7 cents: = 5 cents 1 cents 1 cents
8 cents: = 5 cents 2 cents 1 cents
```

Example:

An example of a solution execution:

Look at the following program snippet:

```
public class Coins {

    static int[] coins   = { 1, 1, 2, 5, 25, 25, 25 };
    static int[] toPay   = { 0, 1, 4, 5, 7, 8 };
    ...
}
```

My program produces the following output:

```
% java Coins
0 cents:            can not be paid
1 cents:            yes; used coins = 1 cents
4 cents:            yes; used coins = 2 cents 1 cents 1 cents
5 cents:            yes; used coins = 5 cents
7 cents:            yes; used coins = 5 cents 1 cents 1 cents
8 cents:            yes; used coins = 5 cents 2 cents 1 cents
```

Your program has to function correctly you change only the amount and values of the available coins.
If I change the values to, and this is the only change:

```
    static int[] coins = { 1, 5, 25, 25, 25 };
    static int[] toPay = { 0, 2, 30, 75 };
```

My program produces the following output::

```
% java Coins
0 cents:            can not be paid
2 cents:            no; can not be paid with the following sequence of coins: [1, 5, 25, 2
30 cents:           yes; used coins = 25 cents 5 cents
75 cents:           yes; used coins = 25 cents 25 cents 25 cents
```

Idea for a Solution:

It might help to understand how to find all possible combinations of *n* items. For a moment assume the set is { 1, 2, 3 }.

{ } ∪ { 1 } = { 1 }
{ 1 } ∪ { 2 } = { 1, 2 }
{ 1 } ∪ { 3 } = { 1, 3 }
and so on. This is the staring point of a recursive solution.

Requirements:

• You have to name your program *Coins.java*

• Your program must produce the correct output, if the only the amount and values of the available coins are changed

Submission:

Submit your files via myCourses.

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
 1      /*
 2       * Coinis = { c_1, c_2, ... c_n }
 3       * k = | Coins | = (2 ^ n)  - 1
 4       */
 5
 6      import java.util.Arrays;
 7
 8
 9      public class Coins {
10
11          static int[] myCoins = { 0, 1, 1, 2, 5, 25, 25, 25 };
12          static int soManyCoins = myCoins.length;
13          static int[] toPay = { 1, 4, 5, 7, 8};
14          static int setSize          = (int)Math.pow(2,  myCoins.length );
15
16          private static void sortCoins() {
17              for (int index = 0; index < myCoins.length - 1; index++)     {
18                      for (int walker = 0; walker < myCoins.length - 1; walker++)  {
19                              if ( myCoins[walker] > myCoins[walker+1] )        {
20                                      int tmp = myCoins[walker];
21                                      myCoins[walker] = myCoins[walker + 1];
22                                      myCoins[walker+1] = tmp;
23                              }
24                      }
25              }
26          }
27          private static String printUsedCoins(int value) {
28              String returnValue = "";
29              for ( int index = soManyCoins; index >= 0 ; index --)   {
30                      if ( ( ( 1 << index ) & value ) == ( 1 << index ) )
31                              returnValue += myCoins[index] + " cents ";
32
33              }
34              if ( returnValue == "" )
35                      returnValue = "empty set";
36              return returnValue;
37          }
38
39          private static int soManyBitsAre1(int value) {
40              int returnValue = 0;
41              for ( int index = soManyCoins; index >= 0 ; index --)   {
42                      if ( ( ( 1 << index ) & value ) == ( 1 << index ) )
43                              returnValue ++;
44
45              }
46              return returnValue;
47          }
```

```
48
49          private static int calculteSumForThisSet(int value) {
50              int sum = 0;
51              for ( int index = soManyCoins - 1; index >= 0 ; index --)    {
52                      if ( ( ( 1 << index ) & value ) == ( 1 << index ) )     {
53                              sum += myCoins[index];
54                      }
55
56              }
57              return sum;
58          }
59          private static void  testIfaCombinationForThisSumExist(int thisSum) {
60              boolean foundAset       = false;
61              int largestSetSoFar     = 0;
62
63              int index = 0;                  // see comment in loop
64
65              if ( thisSum == 0 )     {
66                      System.out.println("0 cents:\t\tcan not be paid");
67              } else {
68
69                      while  ( index < setSize ) {
70                              int sum = calculteSumForThisSet(index);
71                              if ( ( thisSum == sum ) )       {
72                                      if ( soManyBitsAre1(largestSetSoFar) < soManyBitsA
73                                              largestSetSoFar = index;
74                              }
75                              index ++;
76                      }
77                      if ( soManyBitsAre1(largestSetSoFar) > 0 )
78                              System.out.println(thisSum + " cents:   "       +
79                              "\tyes; used coins = "  + printUsedCoins(largestSetSoFar)
80                      else
81                              System.out.println(thisSum + " cents:   "       +
82                              "\tno; can not be paid with the following sequence of coin
83              }
84          }
85          public static void main( String[] arguments ) {
86              // sortCoins();
87              for ( int index = 0; index < toPay.length; index ++ )
88                      testIfaCombinationForThisSumExist(toPay[index]);
89          }
90      }
```

Source Code: Src/21/CoinsSol.java

1.6. Homework 1.3 (10 Points)

Objective: Designing and implementing a simple algorithm and design a testing strategy.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to design a program which finds all numbers between 0, and 10000 who have the following property: A number n exists, so such the sum of each digit ^n is equal to the number. The number is a sum of 'nth' power of each digit of a n digit number is equal to n digit of a number

An example of a number which has this property is 153.

```
1^3 + 5^3 + 3^3  = 1 + 125 + 27 = 153.
```

1 also has this property.

Explanation:

This homework should be done using the following classes:
String (https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/String.html)
Integer (https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Integer.html)
Math (https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/Math.html)

You should solve this homework using as much of the functionality provided by the classes mentioned above.

Your Work:

You should develop this program in stages. Start with the methods you can implement and test first. Develop a test scenario before you develop the method.

My solution produces the following output for the above variables.

```
% java Numbers
  1     ==      1     has the desired property
        1 ^ 1
  2     ==      2     has the desired property
        2 ^ 1
  3     ==      3     has the desired property
        3 ^ 1
  4     ==      4     has the desired property
        4 ^ 1
  5     ==      5     has the desired property
        5 ^ 1
  6     ==      6     has the desired property
        6 ^ 1
  7     ==      7     has the desired property
        7 ^ 1
  8     ==      8     has the desired property
        8 ^ 1
  9     ==      9     has the desired property
        9 ^ 1
  153   ==      153   has the desired property
```

```
     1 ^ 3 + 5 ^ 3 + 3 ^ 3
 370  ==      370  has the desired property
     3 ^ 3 + 7 ^ 3 + 0 ^ 3
 371  ==      371  has the desired property
     3 ^ 3 + 7 ^ 3 + 1 ^ 3
 407  ==      407  has the desired property
     4 ^ 3 + 0 ^ 3 + 7 ^ 3
```

Your solution should produce a similar output. You output should be adjusted similar to my output.

Requirements:

You have to name the file *Numbers.java*.

Idea for a Solution:

No hint

Submission:

Submit your files via myCourses.

Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
 1      // andersons numbers
 2      import java.lang.Math;
 3
 4      public class Numbers {
 5
 6          final static int MINIMUM = 1;       // 0 is excluded
 7          final static int MAXIMUM = 100000000;
 8          static int[]  theDigits;
 9
10
11          private static void printDigits()   {
12                  for ( int index = 0; index < theDigits.length; index ++ )
13                          System.out.println(index + ":= " + theDigits[index]);
14          }
15          private static void extractDigits(int thisNumber) {
16              theDigits = new int[ ( "" + thisNumber ).length() ];
17              int index = 0;
18              if ( theDigits.length > 1 )     {
19                      do {
20                              theDigits[index++ ] = thisNumber % 10;
21                              thisNumber = thisNumber / 10;
22                      } while  ( thisNumber % 10 != thisNumber        );
23              }
24              theDigits[index++] = thisNumber;
25          }
26          private static boolean includesOnlyZerosAndOnes()   {
27              boolean rValue = true;
28              for ( int index = 0; rValue && ( index < theDigits.length); index ++ )
29                      rValue &= ( theDigits[index] <= 1 );
30              }
```

```
31              return rValue;
32          }
33          private static void printResult(int thisNumber, int sumOfPower, int power) {
34              String format = "%" + ( theDigits.length + 2 ) + "d\t%s\t%" + ( theDigits.
35              System.out.printf(format, thisNumber, " == ", sumOfPower, " has the desire
36              System.out.print("      ");
37              for ( int index = 0; index < theDigits.length; index ++ )       {
38                      System.out.print(theDigits[theDigits.length - index - 1] + " ^ " +
39                      if ( index < theDigits.length - 1 )
40                              System.out.print(" + " );
41              }
42              System.out.println();
43          }
44          // the 'do while' loop will never terminate, if the number includes only 0's a
45          private static void testNotZeroAndOnes(int thisNumber) {
46              int power = 0;
47              int sumOfPower = 0;
48              do {
49                      sumOfPower = 0;
50                      for ( int index = 0; index < theDigits.length; index ++ )       {
51
52                              sumOfPower += Math.pow( (double)theDigits[index], (double)
53                      }
54                      power ++;
55              } while  ( sumOfPower <  thisNumber );
56
57              if ( sumOfPower == thisNumber )
58                      printResult(thisNumber, sumOfPower, power - 1);
59          }
60
61          private static void hasProperty(int thisNumber) {
62              extractDigits(thisNumber);
63
64              if (!  includesOnlyZerosAndOnes() )
65                      testNotZeroAndOnes(thisNumber);
66          }
67          private static void findNumbersWithTheseProperties() {
68              for ( int index = MINIMUM; index < MAXIMUM; index ++ )
69                      hasProperty(index);
70          }
71
72          public static void main( String[] args ) {
73              new Numbers().findNumbersWithTheseProperties();
74          }
75      }
76
77
```

Source Code: Src/21/NumbersSol.java