Problem 7.3: Find smallest number that is evenly divisible by all of the numbers 1 through 20.

Source code:

```
/*
 *  ====================================================================
 *  MultipleDivisors.java: Find the smallest number that is evenly
 *  divisible by all of the integers 2 through 20.
 *
 *  Written by: Mark Austin                                November 2009
 *  ====================================================================
 */

import java.lang.Math;

public class MultipleDivisors {
   public static void main ( String args[] ) {
      boolean foundNo = false;
      int iX = 0;
      while ( foundNo == false ) {
          iX = iX + 20;
          if ( iX % 20 == 0 && iX % 19 == 0 && iX % 18 == 0 &&
               iX % 17 == 0 && iX % 16 == 0 && iX % 15 == 0 &&
               iX % 14 == 0 && iX % 13 == 0 && iX % 12 == 0 &&
               iX % 11 == 0 ) {
               System.out.printf("iX = %10d\n", iX );
               foundNo = true;
          }
      }
   }
}
```

Output:

```
prompt >> java MultipleDivisors
iX =  232792560
prompt >>
```

Problem 7.4: Racetrack

Source code:

```
/*
 *  ====================================================================
 *  Racetrack.java: Compute distance around a running track.
 *
 *  Written By: Mark Austin                                  October 2005
 *  ====================================================================
 */

package basics;

import java.lang.Math;
```

```
import java.util.*;
import java.io.*;
import java.text.*;

public class RaceTrack {
   public static void main( String args[] ) {
      double dRunningDistance;

      // Print header ...

      System.out.println(" X(m) | Distance (m)");
      System.out.println("--------------------");

      for ( double dX = 0.0; dX <= 1.0; dX = dX + 0.2 ) {

          // Compute running distance
          dRunningDistance = 400.0 + 2*Math.PI*dX;

          // Format "X" to one decimal place of accuracy ...

          DecimalFormat formatX = new DecimalFormat( "0.0" );
          String output1 = formatX.format( dX );

          // Format "Running Distance " to two decimal places ...

          DecimalFormat formatR = new DecimalFormat( "00.00" );
          String output2 = formatX.format( dRunningDistance );

          // Print output ....

          System.out.println( output1 + " m       " + output2 + " m" );
      }
      System.out.println("--------------------");
    }
}
```

Note: Of course, now you can simplify the code with formatted output via System.out.printf().

Output: Using ant:

```
prompt >> ant racetrack
Buildfile: /Users/austin/ence688r.d/java-code-basics/build.xml

basic07:
     [java]  X(m) | Distance (m)
     [java] --------------------
     [java] 0.0 m       400.0 m
     [java] 0.2 m       401.3 m
     [java] 0.4 m       402.5 m
     [java] 0.6 m       403.8 m
     [java] 0.8 m       405.0 m
     [java] 1.0 m       406.3 m
     [java] --------------------

BUILD SUCCESSFUL
Total time: 3 seconds
prompt >>
```

Problem 7.9: Compute and print primes up to 1,000.

Source code: For completeness, here is the source code to EulerMath.java:

```
import java.lang.Math;

public class EulerMath {
   public static boolean isPrime(long num){
      if (num < 2 || (num % 2 == 0 && num != 2))
          return false;

      for (int i = 3; i <= Math.sqrt(num); i += 2 )
           if (num % i == 0)
               return false;

      return true;
   }
}
```

And here is a program Prime to compute and print the primes:

```
/*
 * ============================================================
 * Prime.java: Compute and print prime numbers ...
 * ============================================================
 */

public class Prime {
   public static void main(String[] args) {
      int counter = 0;

      System.out.println("==========================================");
      System.out.println("Welcome to the prime number " + "generator!");
      System.out.println("The following are the prime numbers " +
                         "between 1 and 1000!");
      System.out.println("==========================================");

      for (int i = 1; i <= 1000 ; i += 1) {
         if (EulerMath.isPrime(i) == true){
             System.out.print(i + "  ");
                counter += 1;
             if (counter % 10 == 0)
                System.out.println("");
         }
      }
      System.out.println("");
  }
}
```

Output: The abbreviated output is as follows:

```
========================================
Welcome to the prime number generator!
The following are the prime numbers between 1 and 1000!
========================================
2        3        5        7        11       13
17       19       23       29       31       37
41       43       47       53       59       61

... lines of output removed ....
```

```
919      929      937      941      947      953
967      971      977      983      991      997
```

Problem 7.13: Leibnez's series summation

Source code:

```
/*
 *  ================================================================
 *  LeibnezSeries.java: This program sums terms in the Leibnez series
 *
 *      pi/4 = 1 - 1/3 + 1/5 - 1/7 .....
 *
 *  to 1000 terms using:
 *
 *      1. A simple looping construct.
 *      2. An array of series coefficient, which are then summed.
 *
 *  Each summation is compared to the theoretical limit, pi/4.
 *  ================================================================
 */

import java.lang.Math;
import java.util.*;
import java.io.*;
import java.text.*;

public class LeibnezSeries {

   public static void main( String args[] ) {
       double dX, dTotal;
       double dSum;

       // Print welcome message

       System.out.println("Leibnez Series summation");
       System.out.println("-----------------------------");
       double dTheoreticalSum = Math.PI/4.0;
       System.out.println("Theoretical Summation = " + dTheoreticalSum );

       // Compute series summation with simple loop...

       dSum = 0.0;
       for( int i = 1; i <= 1000; i = i + 1) {
            if( i%2 == 1 )
                dSum = dSum + 1.0/(2*i-1);
            else
                dSum = dSum - 1.0/(2*i-1);
       }

       System.out.println("");
       System.out.println("Method 1: Simple Loop");
       System.out.println("-----------------------------------");
       System.out.println("Summation      = " + dSum) ;
       System.out.println("Absolute Error = " + (dSum - dTheoreticalSum) ) ;

       // Now store series terms in an array before computing summation...
```

```
        double dTerm[] = new double[1000];
        for( int i = 1; i <= 1000; i = i + 1) {
             if( i%2 == 1 )
                 dTerm[i-1] = 1.0/(2*i-1);
             else
                 dTerm[i-1] = -1.0/(2*i-1);
        }

        // Walk along array element and sum terms ....

        dSum = 0.0;
        for( int i = 1; i <= 1000; i = i + 1)
             dSum = dSum + dTerm[i-1];

        System.out.println("");
        System.out.println("Method 2: Array Storage + Summation");
        System.out.println("-----------------------------------");
        System.out.println("Summation      = " + dSum) ;
        System.out.println("Absolute Error = " + (dSum - dTheoreticalSum) ) ;

   }
}
```

Output:

```
prompt >> java LeibnezSeries
Leibnez Series summation
------------------------------
Theoretical Summation = 0.7853981633974483

Method 1: Simple Loop
-----------------------------------
Summation      = 0.7851481634599485
Absolute Error = -2.4999993749974525E-4

Method 2: Array Storage + Summation
-----------------------------------
Summation      = 0.7851481634599485
Absolute Error = -2.4999993749974525E-4
prompt >>
```

Problem 7.17: Systematic treatment of overflows, underflows and divide by zero.

Source code: Here is the source code for an approach that detects the problem before it occurs:

```
/*
 * ====================================================================
 * FunctionEvaluation1.java: This program evaluates and prints values
 * for a function y(x), x ranging from -4 to 10 in increments of 0.2.
 *
 * Three trouble spots need handling:
 *
 * 1. At x = 0, the expression x/sin(x) evaluates to not-a-number.
 * 2. At x = 2, there is a divide by zero.
 * 3. The increment 0.2 cannot be stored exactly inside the computer.
 *    Hence, tests for relational equality should be based on tolerances.
 *
```

```
 * Strategy:
 *
 * This program tests for x = 0 and x = 2 before y(x) is actually
 * evaluated, and then a suitable error message is printed.
 *
 * By: Mark Austin                                          November 2007
 * ========================================================================
 */

import java.lang.Math;
import java.util.*;
import java.io.*;
import java.text.*;

public class FunctionEvaluation1 {
    public static void main( String args[] ) {
         double dY, dX;

         // Print header and setup table...

         System.out.println("");
         System.out.println("     x | Function y(x)");
         System.out.println("-----------------------");

         // Loop over range of x values....

        dX = -4.0;
        while ( dX <= 10.0 ) {

           // Detect and handle error conditions apriori.
           // Otherwise evaluate and print x, y(x).

           if ( Math.abs ( dX ) < 0.0005 )
             System.out.println("   0.0    Not-a-Number");
           else if ( Math.abs ( dX - 2.0 ) < 0.0005 )
             System.out.println("   2.0  Divide by zero");
           else {
             dY = (Math.pow (dX,4.0) + (dX/Math.sin(dX)))/(dX-2.0);
             System.out.printf("%6.1f      %10.2f\n", dX, dY );
           }

           // Increment value of dX.

           dX = dX + 0.2;
         }
    }
}
```

And here is the abbreviated code for post detection:

```
/*
 * ========================================================================
 * FunctionEvaluation2.java: This program evaluates and prints values
 * for a function y(x), x ranging from -4 to 10 in increments of 0.2.

 ... comment code removed ...

 * Strategy:
 *
 * This program evaluates y(x), tests for error conditions in the result,
```

```
 * and then prints an appropriate message.
 *
 * Note: Program demonstrates use of special error constants, Double.NaN,
 *       Double.NEGATIVE_INFINITY and Double.POSITIVE_INFINITY.
 *
 * By: Mark Austin                                          November 2007
 * ========================================================================
 */

import java.lang.Math;
import java.util.*;
import java.io.*;
import java.text.*;

public class FunctionEvaluation2 {
    public static void main( String args[] ) {
         double dY, dX;

         // Print header and setup table...

         System.out.println("");
         System.out.println("     x | Function y(x)");
         System.out.println("----------------------");

         // Loop over range of x values....

        dX = -4.0;
        while ( dX <= 10.0 ) {

           // Evaluate y(x) ....

              dY = (Math.pow (dX,4.0) + (dX/Math.sin(dX)))/(dX-2.0);

           // Detect and handle error conditions apriori.
           // Otherwise evaluate and print x, y(x).

           if ( dY == Double.POSITIVE_INFINITY )
             System.out.printf("%6.2f      Positive Infinity\n", dX );
           else if ( dY == Double.NEGATIVE_INFINITY )
             System.out.printf("%6.2f      Negative Infinity\n", dX );
           else if ( dY == Double.NaN )
             System.out.printf("%6.2f      Not a Number\n", dX );
           else
             System.out.printf("%6.2f      %10.2f\n", dX, dY );

           // Increment value of dX.

           dX = dX + 0.25;
         }
    }
}
```

Output: Abbreviated output from the first program is as follows:

```
  ... output removed ...

     x | Function y(x)
----------------------
  -4.0          -41.79
  -3.8          -33.25
```

```
  ... output removed ...

  -0.3           -0.45
   0.0    Not-a-Number
   0.3           -0.58

  ... output removed ...

   1.8          -44.63
   2.0  Divide by zero
   2.3          114.08

  ... output removed ...

  10.0         1247.70
```

And the second program generates the output:

```
     x | Function y(x)
-----------------------
 -4.00          -41.79
 -3.75          -33.25

  ... output removed ...

 -0.25           -0.45
  0.00             NaN
  0.25           -0.58

  ... output removed ...

  1.75          -44.63
  2.00      Positive Infinity
  2.25          114.08

  ... output removed ...

  9.75         1162.11
 10.00         1247.70
```

Note: In the first version of this question said use a steplength of 0.2. But then the program doesn't work as expected. Why? The increment 0.2 cannot be stored exactly, hence by the time the loop reaches x = 2, the error accumulates enough not to trigger Positive Infinity.

Problem 7.20: Compute force on a yacht sail ....

Source code:

```
/*
 *  ============================================================
 *  WindForce.java: Compute and print force due to gusts of wind.
 *
 *  Written By: Mark Austin                            March, 2002
 *  ============================================================
 */

import java.lang.Math;
```

```
import java.text.*;

public class WindForce {

   // Constructor method.

   public WindForce() {};

   // main method : this is where the program execution begins.

   public static void main ( String [] args ) {

       // Create WindForce object .....

       WindForce windforce = new WindForce();

       // Print windforce heading

       System.out.println("     Time      Force");
       System.out.println("(seconds)       (kN)");
       System.out.println("====================");

       // Compute and print wind force

       windforce.computeWindForce();

   }

   // Method that computes the wind force.

   public void computeWindForce() {
      float fT, fTime, fWindForce;

      for ( fTime = 0.0f; fTime <= 3.0f; fTime = (float) (fTime + 0.25 )) {
            fT = (float) (fTime - Math.floor( fTime ));

            if (fT <= 0.3 )
               fWindForce = (float) (4.0 + 15.0*fT - 135.0*Math.pow(fT,
3.0));
            else
               fWindForce = (float) ((731.0 - 171*fT)/140.0);

            printWindForce( fTime, fWindForce );
      }
   }

   // Print wind force.

   public void printWindForce( float fTime, float fWindForce ) {

      DecimalFormat formatOutput = new DecimalFormat ("0.00");
      String output1 = formatOutput.format( fTime );
      String output2 = formatOutput.format( fWindForce );

      System.out.println("     " + output1 + "       " + output2 );
   }
}
```

Output:

```
prompt >> java WindForce
     Time      Force
(seconds)       (kN)
====================
     0.00       4.00
     0.25       5.64
     0.50       4.61
     0.75       4.31
     1.00       4.00
     1.25       5.64
     1.50       4.61
     1.75       4.31
     2.00       4.00
     2.25       5.64
     2.50       4.61
     2.75       4.31
     3.00       4.00
prompt >>
```
