All of my Project-Euler code is available at http://github.com/fhinson/Project-Euler **Francis Hinson**

Problem 1

For this problem, using two simple for loops, I added all the numbers under 1000 that are multiples of 3 or 5 to a variable, and then printed that variable.

```
1
2  public class Problem1 {
3
4          public static void main(String[] args) {
5                  int sum = 0;
6                  for (int i = 1; i < 1000; i++){
7                          if (i % 3 == 0 || i % 5 == 0)
8                                  sum+=i;
9                  }
10                 System.out.println("The sum of all natural numbers below one thousand that are multiples of 3 or 5 is " + sum);
11         }
12
13 }
```

Problem 2

For this problem, I created an ArrayList of Fibonacci numbers and using a for-loop, I incremented the sum of the even ones using a simple conditional statement.

```
1  import java.util.ArrayList;
2
3  public class Problem2 {
4
5          public static void main(String[] args) {
6                  ArrayList<Integer> Fibonacci = new ArrayList<Integer>();
7                  Fibonacci.add(0); Fibonacci.add(1);
8
9                  long sum = 0;
10                 int j = 2;
11
12                 while(Fibonacci.get(j-2)+Fibonacci.get(j-1) < 4000000){
13                         Fibonacci.add(Fibonacci.get(j-2)+Fibonacci.get(j-1));
14                         j++;
15                 }
16
17                 for (int i = 0; i < Fibonacci.size(); i++){
18                         if (Fibonacci.get(i) % 2 == 0)
19                                 sum+=Fibonacci.get(i);
20                 }
21
22                 System.out.println("The sum of even valued Fibonacci Terms less than 4 million is " + sum);
23         }
24
25 }
```

Problem 3

For this problem, I defined the number 600851475143 as a long, and then I incremented from 1 to that number and checked for prime factors, adding them to an arraylist. I then printed the greatest element of the arraylist.

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


public class Problem3 {

	public static void main(String[] args) {
		ArrayList<Long> Primes = new ArrayList<Long>();
		for (long i = 1; i <= 600851475143L; i++){
			if(600851475143L % i == 0){
				if(primecheck(i)==true && i!= 1){
					Primes.add(i);
					System.out.println("Adding " + i);
				}
			}
		}
		System.out.println("The greatest prime factor is " + Collections.max(Primes));
	}

	public static boolean primecheck(long x){
		for (int counter = 2; counter < x; counter++){
			if (x % counter == 0)
				return false;
		}
		return true;
	}
}
```

Problem 4

For this problem, I used a method to break a number down into an array of digits. I then had a method that evaluated to see whether that number was indeed a palindrome. I added all the palindromes to an ArrayList, and I printed the maximum value.

```java
public class Problem4 {

	public static void main(String[] args) {
		ArrayList<Integer> Palos = new ArrayList<Integer>();
		for (int i = 100; i<=999; i++){
			for (int j = 100; j <= 999; j++){
				int k = i*j;
				if (Integer.toString(k).length() == 6){
					if (evaluator(k) == true){
						Palos.add(k);
					}
				}
			}
		}
		System.out.println("The largest palindrome made from the product of two 3-digit numbers " + Collections.ma
	}

	public static boolean evaluator(int x){
		int[] list = lister(convert(x));
		boolean a = false;
		int check = 0;
		for (int i = 0; i<=2; i++){
			if (list[i] == list[5-i]){
				check++;
			}
		}
		if (check == 3)
			a = true;
		return a;
	}

	public static String convert (int x) {
		return Integer.toString(x);
	}

	public static int[] lister(String x){
		int z = Integer.parseInt(x);
		int y = x.length() - 1;

		int [] array = new int[y+1];

		for (int counter = y; counter >= 0; counter--) {

			double a = z / (Math.pow(10, counter));

			int b = (int)(Math.ceil(a) - 1);
			if (counter == 0){
				b = b+1;
			}
			array[y-counter]=b;
			z = (int)(z - (b*Math.pow(10, counter)));
		}
		return array;
```

Problem 5

For this problem, I simply searched for the smallest number evenly divisible by 1-20 with a conditional statement.

```
public class Problem5 {

	public static void main(String[] args) {
		for (int i = 1; i<= 1000000000; i++){
			if (i%1 == 0 && i%2 == 0 && i%3 == 0 && i%4 == 0 && i%5 == 0 && i%6 == 0 && i%7 == 0 && i%8 == 0 &&
				System.out.println("The answer is " + i);
				break;
			}
		}
	}

}
```

Problem 6

For this problem I simply dealt with two for-loops, one which calculated the sum of the squares and another which calculated the square of the sums. I then took the difference between them to arrive at the answer.

```
public class Problem6 {

	public static void main(String[] args) {
		int numsum = 0;
		for (int i = 0; i<=100; i++){
			numsum+=i;
		}
		numsum = numsum*numsum;

		int squaresum = 0;
		for (int i = 0; i<=100; i++){
			squaresum+=(i*i);
		}

		System.out.println("The difference between the sum of the squares and the square of the sum is " + (numsum-sc
	}

}
```

Problem 7

For this problem, I simply used a method that checks whether a number is prime, and looped through a lot of numbers until my counter arrived at 10001.

```java

public class Problem7 {


	public static void main(String[] args) {
		int counter = 0;
		int count = 2;
		while (counter != 10001){
			if (primecheck(count) == true){
				counter++;
			}
			count++;
		}
		System.out.println("The 10001st prime number is " + (count-1));
	}

	public static boolean primecheck(long x){
		for (int counter = 2; counter < x; counter++){
			if (x % counter == 0)
				return false;
		}
		return true;
	}

}
```

Problem 8

For this problem, I put the 1000 digit number into an array of digits, and I added the products of 5 consecutive numbers through a for loop to an arraylist. I then returned the maximum value of that arraylist.

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

public class Problem8 {

	public static void main(String[] args){
		int[] longarray = {7,3,1,6,7,1,7,6,5,3,1,3,3,0,6,2,4,9,1,9,2,2,5,1,1,9,6,7,4,4,2,6,5,7,4,7,4,2,3,5,5,3,4,9,1,
		int product = 1;
		ArrayList<Integer> products = new ArrayList<Integer>();
		for (int i = 0; i < longarray.length; i++){
			if (i < 996){
				product = longarray[i] * longarray[i+1] * longarray[i+2] * longarray[i+3] * longarray[i+4];
			}
			products.add(product);
			product = 1;
		}
		System.out.println("The greatest product is " + Collections.max(products));

	}

}
```

Problem 9

For this problem, I used a double for loop and incremented through values until I found the Pythagorean triple that equated to 1000.

```
public class Problem9 {

	public static void main(String[] args) {
		for (double i = 1; i <500; i++){
			for (double j = 1; j<500; j++){
				if (checker((i*i)+(j*j)) == true && i + j + sqrt((i*i)+(j*j)) == 1000){
					System.out.println("The pythagorean triple is " + i + ", " + j + ", " + sqrt((i*i)+(j
					System.out.println("The product is " + i*j*(sqrt((i*i)+(j*j))));
					break;
				}
			}
		}
	}

	public static boolean checker(double x){
		boolean a = false;
		if (Math.pow(x, 0.5) - Math.ceil(Math.pow(x, 0.5)) == 0)
			a = true;
		return a;
	}

	public static double sqrt(double x){
		return Math.pow(x, 0.5);
	}

}
```

Problem 10

For this problem, I had a simple method to check if the number was prime. Then, I incremented through numbers under 2million, adding the primes along the way. The result was simply the total.

```
public class Problem10 {


	public static void main(String [] sdf){
		long sum = 0;
	for (int i = 2; i<2000000; i++){
		if(primecheck(i)==true){
			sum+=i;
			System.out.println(i);
		}
	}
	System.out.println("The sum of primes below 2million is " + sum);
	}

	public static boolean primecheck(long x){
		for (int counter = 2; counter < x; counter++){
			if (x % counter == 0)
				return false;
		}
		return true;
	}

}
```

Problem 11

For this problem, I put all of the numbers into an array and started analyzing groups of 4 using for loops. I had 4 distinct conditionals to handle left, right, diagonal right, and diagonal left.

```
ArrayList<Integer> sums = new ArrayList<Integer>();

for (int i = 0; i < list.length; i++){
        if(i < list.length-3)
                sums.add(list[i]*list[i+1]*list[i+2]*list[i+3]);
        if(i < 340)
                sums.add(list[i]*list[i+20]*list[i+40]*list[i+60]);
        if(i < 337)
                sums.add(list[i]*list[i+21]*list[i+42]*list[i+63]);
        if(i < 340)
                sums.add(list[i]*list[i+19]*list[i+38]*list[i+57]);
}

System.out.println("Greatest is " + Collections.max(sums));

}
```

Problem 12

For this problem, I had a method which created my triangular numbers. Then I had another method which calculated the number of divisors a number has. I incremented through the triangular numbers using a for loop until I found the one with the number of divisors over 500.

```
public static void main (String[] args) {

        ArrayList<Long> triangle = triangle(5);
        for (int i = 199999; i < 200000; i++) {
                System.out.println(triangle.get(i) + ":" + divisor(triangle.get(i)));
                if (divisor(triangle.get(i)) == 500) {
                        System.out.println("The first triangle number that has over 500 divisors is " + triangle.g
                break;
                }
        }
}

private static ArrayList<Long> triangle(int x) {

        ArrayList<Long> triangle = new ArrayList<Long>();
        triangle.add((long) 1);
        long sum = 0;
        for(long i = 1; i < 200000; i++) {
                sum += i;
                triangle.add(sum);
                //System.out.println(triangle.get(i));
        }
        return triangle;
}

private static int divisor(long a) {
        int count = 0;
        for(int i = 1; i <= a; i++) {
                if (a % i == 0)
                        count += 1;
        }
        return count;
}
```

Problem 13

For this problem, I simply summed the large numbers using the Java BigInteger class. I then found the first 10 digits of the sum easily.

```
public static BigInteger sum(ArrayList<BigInteger> list) {
      BigInteger sum= new BigInteger("0");
      for (BigInteger i:list)
          sum = sum.add(i);
      return sum;
}
```

Problem 16

For this problem, I contained 2^1000 in a BigInteger, and then I broke the number down into an array of digits. Using a for-loop, I easily summed the digits.

```
public static void main(String[] args) {
        BigInteger value= new BigInteger("10715086071862673209484250490600018105614048117055336074437503883703510511

        ArrayList<Integer> list = new ArrayList<Integer>();
        BigInteger ten = new BigInteger("10");
        while (!value.equals(BigInteger.ZERO))
        {
            list.add(0, value.mod(ten).intValue());
            value = value.divide(ten);
        }

        int sum = 0;

        for (int i = 0; i < list.size(); i++){
                sum+=list.get(i);
        }

        System.out.println("The sum is " + sum);

}
```

Problem 17

For this problem, I essentially created a small dictionary for the program to refer to of numbers in numerical format. I then designed an algorithm to combine certain numericals. For example, it would take 42 and make that 40 + 2. Using a lot of looping structures, I arrived at the total length.

```
80                for (int n = 1; n < numericals.length; n++){
81                        result+=numericals[n];
82                        System.out.println(numericals[n]);
83                }
84                System.out.println("If all the numbers from 1 to 1000 inclusive were written out in words, " + result.length
85
```

Problem 20

For this problem, I stored 100! in a BigInteger and then I broke the number down into an array of digits and simply added the digits using a for loop.

```
for (int i = 0; i < list.size(); i++){
        sum+=list.get(i);
}

System.out.println("The sum is " + sum);
```

Problem 21

For this problem, I created a method to sum the divisors of a number. I then looped through several numbers to find amicable pairs, and I summed them.

```
4  public class Problem21 {
5          public static ArrayList<Integer> divisors(int a){
6                  ArrayList<Integer> divisors = new ArrayList<Integer>();
7                  for (int i = 1; i < a; i++){
8                          if (a % i == 0){
9                                  divisors.add(i);
10                         }
11                 }
12                 return divisors;
13         }
14         public static int sum(ArrayList<Integer> a){
15                 int s = 0;
16                 for (int i = 0; i <a.size(); i++){
17                         s+=a.get(i);
18                 }
19                 return s;
20         }
21         public static boolean check(int a){
22                 int b = sum(divisors(a));
23                 if(sum(divisors(b))==a && a!=b){
24                         return true;
25                 }
26                 else return false;
27         }
28         public static void main(String[] args){
29                 int sum = 0;
30                 for (int i = 1; i < 10000; i++){
31                         if (check(i)==true) {
32                                 System.out.print(i);
33                                 System.out.print(" : " +divisors(i));
34                                 System.out.println();
35                                 sum+=i;
36                         }
37                 }
38                 System.out.println("The sum of the amicable numbers under 10000 is " + sum);
39         }
```

Problem 25

For this problem, I had a method that uses BigIntegers to generate Fibonacci numbers. I then used a forloop to increment through these numbers, parsed them to strings to find their lengths, and found the one with the length of 1000.

```
public class Problem25 {

	public static void main (String[] args) {

		int x = -1;
		for (int i = 1; i < 10000; i++) {
			x++;
			if (fibonacci(i).toString().length() == 1000) {
				System.out.println("First term to contain 1000 digits in the fibonacci sequence is " + x);
				break;
			}
		}
	}

	private static BigInteger fibonacci (int x) {
		BigInteger p1 = new BigInteger("0");
		BigInteger p2 = new BigInteger("1");
		for (int i = 1; i < x; i++) {
			BigInteger savePrev = p1;
			p1 = p2;
			p2 = savePrev.add(p2);
		}
		return p1;
	}
```

Problem 29

For this problem, I simply used a double for loop and stored all of the values of a^b in an ArrayList. I then dumped all of the values of the arraylist into a hashset, which removed duplicates. I then obtained the total number of values.

```
public class Problem29 {

	public static void main(String[] args){
		ArrayList<BigInteger> numbers = new ArrayList<BigInteger>();
		BigInteger hundred = new BigInteger("101");
		 for (BigInteger bi = BigInteger.valueOf(2); bi.compareTo(hundred) < 0; bi = bi.add(BigInteger.ONE)) {
			for (int bn = 2; bn <= 100; bn++){
				numbers.add(bi.pow(bn));
			}
		}

		 HashSet hs = new HashSet();
		 hs.addAll(numbers);
		 numbers.clear();
		 numbers.addAll(hs);

		System.out.println("The total number of distinct numbers is " + numbers.size());
	}

}
```

Problem 30

For this problem, I created a method which breaks a number into digits and sums the fifth power of the digits and determines whether they equal the original number. I then used a for loop to find such numbers, and I took the sum of them.

```
public static void main(String[] args){
        int sum = 0;
        for (int i = 2; i<354294; i++){
                if(lister(i)==true){
                        System.out.println(i);
                        sum+=i;
                }
        }
        System.out.println("The sum of all the numbers that can be written as the sum of fifth powers of their digits
}
```

Problem 33

For this problem, I used a method to break numbers into digits, and then I started removing digits from the fractions and making evaluations using lots of looping and conditionals.

```
for (double i = 10; i <= 99; i++){
        for (double j = 10; j <= 99; j++){
                double[] list1 = list(i), list2 = list(j);
                if (i/j < 1){
                        if(list1[0] != 0 && list1[1] != 0 && list2[0] != 0 && list2[1] != 0){
                                if(list1[1] == list2[0]){
                                        if((i/j) == list1[0] / list2[1]){
                                                numerator.add(i); denominator.add(j);
                                        }
                                }
                                if(list1[0] == list2[1]){
                                        if((i/j) == list1[1] / list2[0]){
                                                numerator.add(i); denominator.add(j);
                                        }
                                }
                        }
                }
        }
}

long product = 1, product2 = 1;
for (int k = 0; k < denominator.size(); k++){
        product*=denominator.get(k);
        product2*=numerator.get(k);
}
long gcd = gcd(product, product2);
System.out.println("The fraction is " + product2/gcd + "/" + product/gcd);
```

Problem 34

For this problem, I broke a number down into an array of digits, and summed the factorials of the number. I looped through several numbers to find my matches, and summed them. I used a recursive method to factorial the numbers.

```
public static void main(String[] args) {
        for (int i = 3; i < 10000000; i++ ){
                if (factorialize(list(i)) == i){
                        System.out.println(i + " is a match!!!");
                }
        }
}
```

Problem 36

For this problem, I converted the number to an array of digits, broke the array in half, checked if the 1st value corresponds with the last and so on until it reaches the middle. If there were matches, it increments a counter, and if the number is indeed a palindrome, the number of matches should be equal to half the length of the number. I applied this in a looping structure to regular numbers and their binary equivalent to find the sum.

```
public class Problem36 {

        public static void main(String[] args) {
                int sum = 0;
                for (int i = 1; i <= 1000000; i++){
                        BigInteger binary = new BigInteger(Integer.toBinaryString(i));
                        if(evaluator(BigInteger.valueOf(i)) == true && evaluator(binary)== true){
                                sum+=i;
                        }
                }
                System.out.println("The sum of all numbers, less than one million, which are palindromic in base 10 and base
        }

        public static boolean evaluator(BigInteger x){
                ArrayList<Integer> foo = list(x);
                Integer[] list = foo.toArray(new Integer[0]);
                int check = 0;
                        for(int i = 0; i <= (list.length/2)-1; i++){
                                if(list[i] == list[list.length-i-1]){
                                        check++;
                                }
                        }
                        if (check == list.length/2)
                                return true;
                return false;

        }
        public static ArrayList<Integer> list(BigInteger value){
                ArrayList<Integer> list = new ArrayList<Integer>();
                BigInteger ten = new BigInteger("10");
                while (!value.equals(BigInteger.ZERO))
                {
                    list.add(0, value.mod(ten).intValue());
                    value = value.divide(ten);
                }
                return list;
        }
}
```

Problem 47

For this problem, I wrote methods to find the number of divisors and whether they are prime. I then used a for loop to find 4 consecutive numbers that have those traits.

```
public class Problem47 {

	public static void main(String[] args) {
		int check = 0;
		for (int i = 1; i < 1000000; i++){

			if(divisor(i) == 4){
				check++;
				System.out.println(i + " is a match! : " + check);
				if (check == 4){
					System.out.println((i-3) + " is the answer!");
					break;
				}
				if (divisor(i+1) != 4){
					check = 0;
				}
			}
		}
	}

	public static boolean primecheck(long x){
		for (int counter = 2; counter < x; counter++){
			if (x % counter == 0)
				return false;
		}
		return true;
	}

	private static int divisor(int a) {
		int count = 0;
		for(int i = 2; i <= a; i++) {
			if (a % i == 0 && primecheck(i) == true)
				count += 1;
		}
		return count;
	}

}
```

Problem 48

For this problem, I simply used the BigInteger class to make this large summation. I then easily obtained the last 10 digits.

```
public static void main(String[] args) {
	BigInteger sum = new BigInteger("0");
	BigInteger thousand = new BigInteger("1000");
	for (int i = 1; i <=1000; i++){
		sum = sum.add(BigInteger.valueOf(i).pow(i));
		//System.out.println(BigInteger.valueOf(i).pow(i));
	}
	System.out.println(sum);

}
```

Problem 50

For this problem, I used a lot of looping structures to search for the longest chain of primes that sum to another prime. I used a counter point system to determine the maximum length.

```
public class Problem50 {
	public static boolean primecheck(long x){
		for (int counter = 2; counter < x; counter++){
			if (x % counter == 0)
				return false;
		}
		return true;
	}
	public static void main(String[] args){
		ArrayList<Integer> primes = new ArrayList<Integer>();
		int sumcheck = 0;
		for (int i = 2; i < 1000000; i++){
			if (primecheck(i)==true){
				primes.add(i);
				System.out.println(i);
				for (int k = 0; k <primes.size()-1; k++){
					for (int j = k; j<primes.size()-1;j++){
						sumcheck+=primes.get(j);
						if(sumcheck == i){
							System.out.println("BINGO !!!!" + i + " with a length of " + (j+1));
							break;
						}
					}
					sumcheck = 0;
				}
				sumcheck = 0;
			}
		}
		System.out.println(primes);
	}

}
```