

# Method Comparison

Steven Hanna

February 2, 2015

## 1 isPrime

The objective of the *isPrime* method is to determine whether or not a specified number was a prime number. The two methods that I will be comparing are those of **Andrew Swinston** and **Mitchell Adair**.

---

```
// PrimeFinder.java
// Andrew Swinston

public class PrimeFinder {
    public static void main(String[] args) {
        System.out.println(isPrime(5));
    }
    public static boolean isPrime(int num) {
        boolean isPrime = true;
        if(num == 1) {
            return false;
        }
        if(num % 2 == 0) {
            isPrime = false;
        }
        if(num == 2) {
            isPrime = true;
        }
        for(int i = 3; i <= (num / 2); i += 2) {
            if(num % i == 0) {
                isPrime = false;
            }
        }
        return isPrime;
    }
}
```

---

---

```

// FindingPrimeNumbers.java
// Mitchell Adair
import java.util.Scanner;

public class FindingPrimeNumbers {

    public static void main(String[] args) {
        //System.out.println( isPrime( 4 ) );
        Scanner input = new Scanner( System.in );
        int numOfPrimes = input.nextInt();
        printPrimes( numOfPrimes );
    }

    public static boolean isPrime( int num ) {
        boolean prime = true;
        for (int i = 2; i<=(num/2); i++) {
            if (num%i == 0) {
                prime = false;
            }
        }
        return prime;
    }
}

```

---

## 1.1 Memory Management

For general calculations of memory management:

Java Type	Bytes Required
boolean	1
byte	1
char	2
short	2
int	4
float	4
long	8
double	8

**primeFinder:**  $1(\text{boolean}) + 4(\text{int}) + 4(\text{int}) = 9(\text{bytes})$

**FindingPrimeNumbers:**  $1(\text{boolean}) + 4(\text{int}) + 4(\text{int}) = 9(\text{bytes})$

In regards to memory management, both of these methods utilize the same amount of memory.

## 1.2 Code Readability

*FindingPrimeNumbers* maintains the clearest, and most concise code. Although this is not a true test for efficiency and accuracy, it allows the programmer to easily make changes without stumbling through many un-needed lines.

Upon reflection, the drawback to both programs, lies in the syntax itself. Both programs assume that **prime** or **isPrime** respectively, to be **true**. Although the program remains operational, exceptions can be triggered that will result in a **true** return statement, when it should have been false. This could drastically alter the results of the program. It is much safer to assume **false**, unless proven otherwise.

## 2 printPrimes

The objective of the *printPrimes* method is to print *n* amount of prime numbers, in a row with only 10 values each. Because **Andrew Swinston** did not include the *printPrimes* statement in his submission, (Fatima Azfar) and **Mitchell Adair's** programs will be used.

---

```
// PrintPrimes.java
// Fatima Azfar
import java.util.Scanner.*;

public class printPrimes {
    public static void printPrimes(int num) {
        int counter = 0;
        int lineTen = 0;
        int testPrime = 2;
        while(counter<num) {
            if(isPrime(testPrime)) {
                if(lineTen<10) {
                    System.out.print(" "+testPrime);
                }
                else {
                    System.out.print("\n "+testPrime);
                    lineTen=0;
                }
                counter++;
                lineTen++;
            }
            testPrime++;
        }
    }

    public static void main(String[]args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter Number");
        int numberOfPrimes= input.nextInt();
        printPrimes(numberOfPrimes);
        input.close();
    }
}
```

```

    }
}



---


// FindingPrimeNumbers.java
// Mitchell Adair
import java.util.Scanner;

public class FindingPrimeNumbers {

    public static void main(String[] args) {
        //System.out.println( isPrime( 4 ) );
        Scanner input = new Scanner( System.in );
        int numOfPrimes = input.nextInt();
        printPrimes( numOfPrimes );
    }

    public static void printPrimes( int num ) {
        int counter = 0, primeTest = 2;
        while (counter < num) {
            if (isPrime(primeTest)) {
                if (counter != num-1) {
                    System.out.print( primeTest + ", " );
                    counter++;
                } else {
                    System.out.print( primeTest + "." );
                    counter++;
                }
            }
            if (counter%10 == 0) {
                System.out.println();
            }
            primeTest++;
        }
    }
}

```

---

## 2.1 Memory Management

For general calculations of memory management:

Java Type	Bytes Required
boolean	1
byte	1
char	2
short	2
int	4
float	4
long	8
double	8

**PrintPrimes:**  $4(int) + 4(int) + 4(int) + 4(int) = 16(bytes)$

**FindingPrimeNumbers**  $4(int) + 4(int) + 4(int) = 12(bytes)$

*FindingPrimeNumbers* is more efficient, only using 12bytes.

## 2.2 Code Readability

Personally, I find that *FindingPrimeNumbers* maintains the clearest, and most concise code. *PrintPrimes* uses an additional counter, cluttering the code, which could ultimately result in future errors.

However, when executing *FindingPrimeNumbers*, a strange result is displayed. The program has each number neatly in a row, however when it has to go to a new line, it seemingly randomly decides an amount of line spaces to execute, resulting in a jumbled mess. Therefore, *PrintPrimes* is the superior algorithm.