

## Lecture 6

### Arrays

If a program has a large number of variables then, in order to get a handle on them, it is usually necessary to index them by means of arrays.

A pair of square brackets, [], is the java notation for an array. For example, if a program were to deal with 10 states, say

```
California, Oregon, Washington, Montana, Idaho,  
Wyoming, Colorado, Arizona, Nevada, Utah
```

then the name of the array might be state and the individual array elements might be state[0], state[1], .. , state[9] where

```
state[0] = California, state[1] = Oregon,  
state[2] = Washington, state[3] = Montana,  
state[4] = Idaho, state[5] = Wyoming,  
state[6] = Colorado, state[7] = Arizona,  
state[8] = Nevada, state[9] = Utah
```

The terminology here is state[] is an array (of strings) of 10 elements. Note that the index runs from 0 to 9, and not from 1 to 10. This is a java convention: the index of an array of n elements runs from 0 to n-1.

There are several ways to declare and allocate space for an array in java. One might give the instructions:

```
String state[]; //declare  
String state[] = new String[10]; // allocate
```

These two instructions can be written on one line as:

```
String state[] = new String[10];
```

Alternatively, the pair of brackets can be added to the type name:

```
String[] state = new String[10];
```

Strings can be of any type, but all elements of a String must be of the same type.

The next example shows one way of declaring, allocating, and defining an array and its elements.

Note that all the individual elements of the String array `state` below must be enclosed in double quote marks (`//1`). In addition `state.length` is the number of elements of the array (`//2`).

```

public class States
{
    public static void main(String args[])
    {
        String state[] = {"California", "Oregon", "Washington",
                          "Montana", "Idaho", "Wyoming",
                          "Colorado", "Arizona", "Nevada", "Utah"}; //1

        int prime[] = {2,3,5,7,11,13,17, 19};

        System.out.println("/n");
        for (int i = 0; i < state.length; i++) //2
        {
            System.out.print("state["+i+"] = "+state[i]+ " ");
            if (i%2 == 1) System.out.println();
        }

        System.exit(0);
    }
}

```

The output is:

```

Command Prompt
Z:\11Pic20\Lecture_6\States>java States

state[0] = California  state[1] = Oregon
state[2] = Washington  state[3] = Montana
state[4] = Idaho  state[5] = Wyoming
state[6] = Colorado  state[7] = Arizona
state[8] = Nevada  state[9] = Utah

Z:\11Pic20\Lecture_6\States>

```

## Testing for Primality

Recall that a positive integer  $\geq 2$  is said to be a prime if it is divisible only by 1 and itself. (By definition, the integer 1 is not considered to be a prime). Thus the first few primes are 2, 3, 5, 7, 11, 13, ..

The simplest way to determine whether a given integer  $n$  is to test to see if it is divisible by any integer  $\leq \sqrt{n}$ . The reasoning behind the test is simple: If  $n$  is not a prime it must be divisible by a prime that is  $\leq \sqrt{n}$ ; it can't have two prime factors that are both greater than  $\sqrt{n}$ .

The next application is based on this idea.

```
public class Primes
{
    public static void main(String args[])
    {
        for (int n = 3; n < 30; n = n +2)
            if (isPrime(n))
                System.out.println( n + " is a prime");
    }

    public static boolean isPrime(int n)
    {
        boolean isprime = true;

        int N = (int)Math.sqrt(n);

        for (int i = 2; i <= N ; i++)
        {
            if (n%i == 0)
            {
                isprime = false;
                break;
            }
        }
        System.out.println("n = " +n+ " i = " + i);
    } // end for

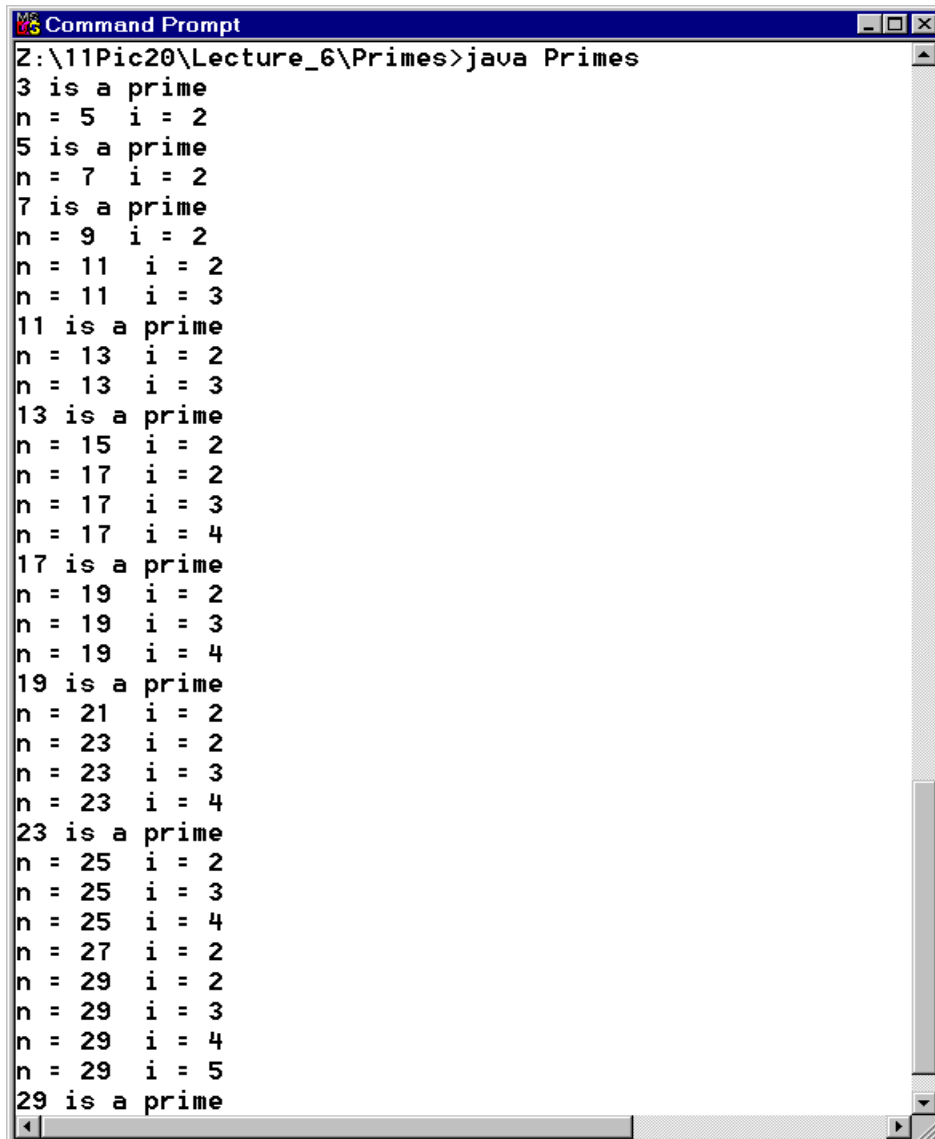
    return isprime;
} // end class Primes
```

Note that we have added a method `isPrime(int n)` to the application `main(String args[])` here.

The output for this application shows how the "for" loop and the **break** work together:

```
int N = (int)Math.sqrt(n);

for (int i = 2; i <= N ; i++)
    if (n%i == 0)
    {
        isprime = false;
        break;
    }
System.out.println("n = " +n+ "  i = " + i);
} // end for
```



```
Command Prompt
Z:\11Pic20\Lecture_6\Primes>java Primes
3 is a prime
n = 5 i = 2
5 is a prime
n = 7 i = 2
7 is a prime
n = 9 i = 2
n = 11 i = 2
n = 11 i = 3
11 is a prime
n = 13 i = 2
n = 13 i = 3
13 is a prime
n = 15 i = 2
n = 17 i = 2
n = 17 i = 3
n = 17 i = 4
17 is a prime
n = 19 i = 2
n = 19 i = 3
n = 19 i = 4
19 is a prime
n = 21 i = 2
n = 23 i = 2
n = 23 i = 3
n = 23 i = 4
23 is a prime
n = 25 i = 2
n = 25 i = 3
n = 25 i = 4
n = 27 i = 2
n = 29 i = 2
n = 29 i = 3
n = 29 i = 4
n = 29 i = 5
29 is a prime
```

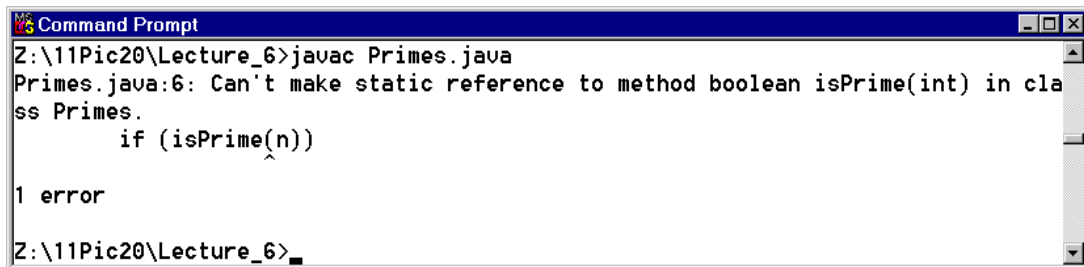
## Variations on "static"

Having checked that `isPrime(int n)` is working we erase the print lines inside it. In addition, to see what will happen, we erase the words "static" from the first line of its definition. The result is:

```
public class Primes
{
    public void main(String args[])
    {
        for (int n = 3; n < 30; n = n + 2)
            if (isPrime(n))
                System.out.println( n + " is a prime");
    }

    public boolean isPrime(int n)
    {
        boolean isprime = true;
        int N = (int)Math.sqrt(n);
        for (int i = 2; i <= N ; i++)
        {
            if (n%i == 0)
            {
                isprime = false;
                break;
            }
        } // end for
        return isprime;
    }
} // end class Primes
```

Then we compile and get an error message:

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the following text:

```
Z:\11Pic20\Lecture_6>javac Primes.java
Primes.java:6: Can't make static reference to method boolean isPrime(int) in class Primes.
    if (isPrime(n))
           ^
1 error
Z:\11Pic20\Lecture_6>_
```

There is another way of getting the program to run. We will return to this point later

## An Array of Primes

Let us return to constructing an array of primes. We shall use the method `isPrime(int n)` to produce an applet that counts and displays the primes that are  $\leq n$ , where  $n$  is a given positive integer. We begin by including `isPrime(int n)` in the first stage of our applet and checking that it works. Note the absence of the word `static` in the definition of `isPrime()`.

```
import javax.swing.*;

public class PrimeArray extends JApplet
{
    public void init()
    {
        String w;
        int n ;

        w = JOptionPane.showInputDialog("Enter an integer n");
        n = Integer.parseInt(w);

        int count = 1;
        for (int j = 3; j <= n; j=j+2)
            if (isPrime(j)) count++;

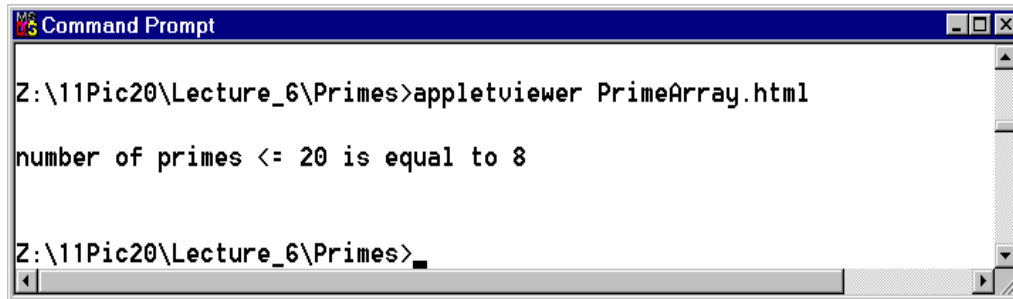
        System.out.print("\nnumber of primes <= " + n);
        System.out.println(" is equal to " + count+ "\n");
    } // end init()

    public boolean isPrime(int n)
    {
        boolean isprime = true;
        int N = (int)Math.sqrt(n);

        for (int i = 2; i <= N ; i++)
        {
            if (n%i == 0)
            {
                isprime = false;
                break;
            }
        } // end for

        return isprime;
    } // end isPrime(int n)
} // end class PrimeArray
```

The output of this for  $n = 20$  verifies that it is functioning:



So, we go on to the problem of constructing the array. As before, old code is in blue; new is in red.

```
import javax.swing.*;

public class PrimeArray extends JApplet
{
    public void init()
    {
        String w;
        int n ;

        w = JOptionPane.showInputDialog("Enter an integer n");
        n = Integer.parseInt(w);

        int count = 1;
        for (int j = 3; j <= n; j=j+2)
            if (isPrime(j)) count++;

        System.out.print("\nnumber of primes <= " +n);
        System.out.println(" is equal to " + count+ "\n");

        int[] pr = new int[count];

        pr[0] = 2;
        int newc = 1;
        for (int j = 3; j <=n ; j = j+2)
            if (isPrime(j))
            {
                pr[newc] = j;
                newc++;
            }

        for(int j = 0; j < count; j++)
        {
            System.out.print(" pr["+j+"] = " + pr[j]);
        }
    }
}
```

```

        if (j%5 == 4) System.out.println();
    } // end for

} // end init()

public boolean isPrime(int n)
{
    boolean isprime = true;
    int N = (int)Math.sqrt(n);

    for (int i = 2; i <= N ; i++)
    {
        if (n%i == 0)
        {
            isprime = false;
            break;
        }
    } // end for

    return isprime;
} // end isPrime(int n)

} // end class PrimeArray

```

In short, once the number of primes  $\leq n$  is determined we set up the array to hold the primes by the line

```
int[] pr = new int[count];
```

The output for  $n = 200$  is:

```

Command Prompt
Z:\11Pic20\Lecture_6\Primes>appletviewer PrimeArray.html
number of primes <= 200 is equal to 46
pr[0] = 2 pr[1] = 3 pr[2] = 5 pr[3] = 7 pr[4] = 11
pr[5] = 13 pr[6] = 17 pr[7] = 19 pr[8] = 23 pr[9] = 29
pr[10] = 31 pr[11] = 37 pr[12] = 41 pr[13] = 43 pr[14] = 47
pr[15] = 53 pr[16] = 59 pr[17] = 61 pr[18] = 67 pr[19] = 71
pr[20] = 73 pr[21] = 79 pr[22] = 83 pr[23] = 89 pr[24] = 97
pr[25] = 101 pr[26] = 103 pr[27] = 107 pr[28] = 109 pr[29] = 113
pr[30] = 127 pr[31] = 131 pr[32] = 137 pr[33] = 139 pr[34] = 149
pr[35] = 151 pr[36] = 157 pr[37] = 163 pr[38] = 167 pr[39] = 173
pr[40] = 179 pr[41] = 181 pr[42] = 191 pr[43] = 193 pr[44] = 197
pr[45] = 199
Z:\11Pic20\Lecture_6\Primes>

```

An aesthetic problem remains: the output looks sloppy. It can be improved by adding appropriate blanks to the output lines. The method for doing this, `blanks(int n)`, is at the end of the complete program for the applet below. It is applied in the printout lines at the end of the `init()` method.

```

import java.awt.*;
import javax.swing.*;

public class PrimeArray extends JApplet
{
    public void init()
    {
        String w;
        int n ;

        w = JOptionPane.showInputDialog("Enter an integer n");
        n = Integer.parseInt(w);

        int count = 1;
        for (int j = 3; j <= n; j=j+2)
            if (isPrime(j)) count++;

        System.out.print("\nnumber of primes <= " +n);
        System.out.println(" is equal to " + count+ "\n");

        int[] pr;
        pr = new int[count];

        pr[0] = 2;
        int newc = 1;
        for (int j = 3; j <=n ; j = j+2)
            if (isPrime(j))
            {
                pr[newc] = j;
                newc++;
            }

        for (int j = 0; j < count; j++)
        {
            System.out.print(blanks(j)+" pr["+j+"] = ");
            System.out .print(blanks(pr[j])+ pr[j]);
            if (j%5 == 4) System.out.println();
        }

    } // end init()

    public boolean isPrime(int n)
    {
        boolean isprime = true;
        int N = (int)Math.sqrt(n);

        for (int i = 2; i <= N ; i++)
        {
            if (n%i == 0)
            {
                isprime = false;
                break;
            }
        } // end for

        return isprime;
    }
}

```

```

} // end isPrime(int n)

public String blanks(int n)
{
    String spaces = "";

    if (0 <= n && n < 10) spaces = " ";
    if (10 <= n && n < 100) spaces = " ";
    return spaces;
}

} // end class PrimeArray

```

```

Command Prompt
Z:\11Pic20\Lecture_6\Primes>appletviewer PrimeArray.html

number of primes <= 600 is equal to 109

pr[0] = 2    pr[1] = 3    pr[2] = 5    pr[3] = 7    pr[4] = 11
pr[5] = 13   pr[6] = 17   pr[7] = 19   pr[8] = 23   pr[9] = 29
pr[10] = 31  pr[11] = 37  pr[12] = 41  pr[13] = 43  pr[14] = 47
pr[15] = 53  pr[16] = 59  pr[17] = 61  pr[18] = 67  pr[19] = 71
pr[20] = 73  pr[21] = 79  pr[22] = 83  pr[23] = 89  pr[24] = 97
pr[25] = 101 pr[26] = 103 pr[27] = 107 pr[28] = 109 pr[29] = 113
pr[30] = 127 pr[31] = 131 pr[32] = 137 pr[33] = 139 pr[34] = 149
pr[35] = 151 pr[36] = 157 pr[37] = 163 pr[38] = 167 pr[39] = 173
pr[40] = 179 pr[41] = 181 pr[42] = 191 pr[43] = 193 pr[44] = 197
pr[45] = 199 pr[46] = 211 pr[47] = 223 pr[48] = 227 pr[49] = 229
pr[50] = 233 pr[51] = 239 pr[52] = 241 pr[53] = 251 pr[54] = 257
pr[55] = 263 pr[56] = 269 pr[57] = 271 pr[58] = 277 pr[59] = 281
pr[60] = 283 pr[61] = 293 pr[62] = 307 pr[63] = 311 pr[64] = 313
pr[65] = 317 pr[66] = 331 pr[67] = 337 pr[68] = 347 pr[69] = 349
pr[70] = 353 pr[71] = 359 pr[72] = 367 pr[73] = 373 pr[74] = 379
pr[75] = 383 pr[76] = 389 pr[77] = 397 pr[78] = 401 pr[79] = 409
pr[80] = 419 pr[81] = 421 pr[82] = 431 pr[83] = 433 pr[84] = 439
pr[85] = 443 pr[86] = 449 pr[87] = 457 pr[88] = 461 pr[89] = 463
pr[90] = 467 pr[91] = 479 pr[92] = 487 pr[93] = 491 pr[94] = 499
pr[95] = 503 pr[96] = 509 pr[97] = 521 pr[98] = 523 pr[99] = 541
pr[100] = 547 pr[101] = 557 pr[102] = 563 pr[103] = 569 pr[104] = 571
pr[105] = 577 pr[106] = 587 pr[107] = 593 pr[108] = 599

Z:\11Pic20\Lecture_6\Primes>

```

Homework:

- 1) Will the method `isPrime(n)` determine if `n` is a prime if `n` is a number less than 2 billion? Why? What if `n` is a number `n` whose size is about 3 billion? Why?
- 2) Suppose that for a given number `n` whose size is about  $10^8$ , `isPrime(n)` is true. How many evaluations of the type `n%k` must be made to determine that `isPrime(n)` is true?
3. Suppose that you wanted to set up an array of all the primes  $\leq 10,000,000$  without actually counting the number of primes  $\leq 10,000,000$ . One way of doing this would be by the instruction

```
int[] prime = new int[10000000];
```

Comment on the sensibility of this.

3. Suppose you wrote

```
int state[] = {California, Texas, Illinois};
```

in a syntactically correct program. What then is California? What sense is there, if any, in the commands

```
California =      ; // define the value of California  
Illinois =       ; // define the value of Illinois
```

```
California*Illinois;
```

4. Given time, would your computer eventually carry out the instruction

```
int count = 0;  
for (int n = 3; n < 1000000000; n = n+2)  
    if isPrime(n) count++;
```

(Assume you have at least a Pentium II machine with 64 Mb of Ram)