

Administrivia

- Please make sure you have obtained a Unix account. If you are a concurrent enrollment student not yet on our lists, please tell a TA so that we can have you added to those eligible to receive an account.
- If you did not complete Lab #1, please try to do so over the weekend (usually, they are due Friday midnight). It is especially important to set up your central repositary.
- If you decide not to take this course after all, please tell CalCentral ASAP, so that we can adjust the waiting list accordingly.
- Those of you on the waiting list should find a lab section that is open, remove yourself from the waiting list, and re-add with this open lab section. The waiting list is processed twice daily.
- HW #0 now up; due next Friday at midnight. You get credit for any submission, but we suggest you give the problems a serious try.

Lecture #2: Let's Write a Program: Prime Numbers

Problem: want java Primes U to print prime numbers through U .

You type: java Primes 101

It types: 2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101

Definition: A *prime* number is an integer greater than 1 that has no divisors smaller than itself other than 1.

Useful Facts:

- $k \leq \sqrt{N}$ iff $N/k \geq \sqrt{N}$, for $N, k > 0$.
- If k divides N then N/k divides N .

So: Try all potential divisors up to and including the square root.

Plan

```
public class Primes {  
    /** Print all primes up to ARGS[0] (interpreted as an  
     * integer), 10 to a line. */  
    public static void main(String[] args) {  
        printPrimes(Integer.parseInt(args[0]));  
    }  
  
    /** Print all primes up to and including LIMIT, 10 to  
     * a line. */  
    private static void printPrimes(int limit) {  
        /*{ For every integer, x, between 2 and LIMIT, print it if  
         isPrime(x), 10 to a line. }*/  
    }  
  
    /** True iff X is prime */  
    private static boolean isPrime(int x) {  
        return /*( X is prime )*/;  
    }  
}
```

Testing for Primes

```
private static boolean isPrime(int x) {  
    if (x <= 1)  
        return false;  
    else  
        return !isDivisible(x, 2); // "!" means "not"  
}
```

```
/** True iff X is divisible by any positive number >=K and < X,  
 * given K > 1. */
```

```
private static boolean isDivisible(int x, int k) {  
    if (k >= x) // a "guard"  
        return false;  
    else if (x % k == 0) // "%" means "remainder"  
        return true;  
    else // if (k < x && x % k != 0)  
        return isDivisible(x, k+1);  
}
```

Thinking Recursively

Understand and check `isDivisible(13,2)` by *tracing one level*.

```
/** True iff X is divisible by
 *  some number >=K and < X,
 *  given K > 1. */
private static boolean isDivisible...
    if (k >= x)
        return false;
    else if (x % k == 0)
        return true;
    else
        return isDivisible(x, k+1);
}
```

Lesson: Comments aid understanding.
Make them *count*!

- Call assigns `x=13, k=2`
- Body has form 'if (`k >= x`) S_1 else S_2 '.
- Since `2 < 13`, we evaluate the first else.
- Check if `13 mod 2 = 0`; it's not.
- Left with `isDivisible(13,3)`.
- Rather than tracing it, instead use the *comment*:
- Since 13 is *not* divisible by any integer in the range 3..12 (and `3 > 1`), `isDivisible(13,3)` must be *false*, and we're done!
- Sounds like that last step begs the question. Why doesn't it?

Iteration

- isDivisible is *tail recursive*, and so creates an *iterative process*.
- Traditional “Algol family” production languages have special syntax for iteration. Four equivalent versions of isDivisible:

```
if (k >= x)
    return false;
else if (x % k == 0)
    return true;
else
    return isDivisible(x, k+1);
```

```
while (k < x) { // !(k >= x)
    if (x % k == 0)
        return true;
    k = k+1;
    // or k += 1, or (yuch) k++
}
return false;
```

```
int k1 = k;
while (k1 < x) {
    if (x % k1 == 0)
        return true;
    k1 += 1;
}
return false;
```

```
for (int k1 = k; k1 < x; k1 += 1) {
    if (x % k1 == 0)
        return true;
}
return false;
```

Using Facts about Primes

- We haven't used the Useful Facts from an earlier slide. Only have to check for divisors up to the square root.
- So, reimplement the iterative version of `isDivisible`:

```
/** True iff X is divisible by some number  $\geq K$  and  $< X$ ,  
 *   given that  $K > 1$ , and that X is not divisible by  
 *   any number  $>1$  and  $<K$ . */  
private static boolean isDivisible(int x, int k) {  
    int limit = Math.round(Math.sqrt(x));  
    for (int k1 = k; k1 <= limit; k1 += 1) {  
        if (x % k1 == 0)  
            return true;  
    }  
    return false;  
}
```

- Why the additional (blue) condition in the comment?

Final Task: printPrimes (Simplified)

```
/** Print all primes up to and including LIMIT. */
private static void printPrimes(int limit) {

}
```


Simplified printPrimes Solution

```
/** Print all primes up to and including LIMIT. */
private static void printPrimes(int limit) {
    for (int p = 2; p <= limit; p += 1) {
        if (isPrime(p)) {
            System.out.print(p + " ");
        }
    }
    System.out.println();
}
```

printPrimes (full version)

```
/** Print all primes up to and including LIMIT, 10 to
 * a line. */
private static void printPrimes(int limit) {
    int np;
    np = 0;
    for (int p = 2; p <= limit; p += 1) {
        if (isPrime(p)) {
            System.out.print(p + " ");
            np += 1;
            if (np % 10 == 0)
                System.out.println();
        }
    }
    if (np % 10 != 0)
        System.out.println();
}
```