Time Measuring of Programs
Analysis of Algorithms

- Dr. 何明昕，He Mingxin, Max
- program06 @ yeah.net
- Email Subject： (L1-|L2-|L3-) + *last 4 digits of ID* + *Name*: *TOPIC*

Your Lab Class

- Sakai: CS203B Fall 2022

数据结构与算法分析B
Data Structures and Algorithm Analysis

CS203B-f22-课程群

该二维码7天内(9月18日前)有效，重新进入将更新

QQ Group for Labs:

813058168

Recap: 3 Dimensions of typical CS Courses

Most of Courses on CS may be organized by a combination of elements from the following 3 dimensions:

THEORY related: Concepts, Models, Maths, Algorithms, Principles, Mechanisms, Methods, … Need to understand Abstract Things

SYSTEM and TOOLS related: HW, Network, OS, PLs, IDEs, DBMS, Clients, Servers, Virtual Machines, Containers in Cloud, … Need to understand, setup and use them properly

DESIGN related: According to Requirements (Problems), need to use Theory and Tools to shape/create/implement/test Solutions!

Put them All Together!

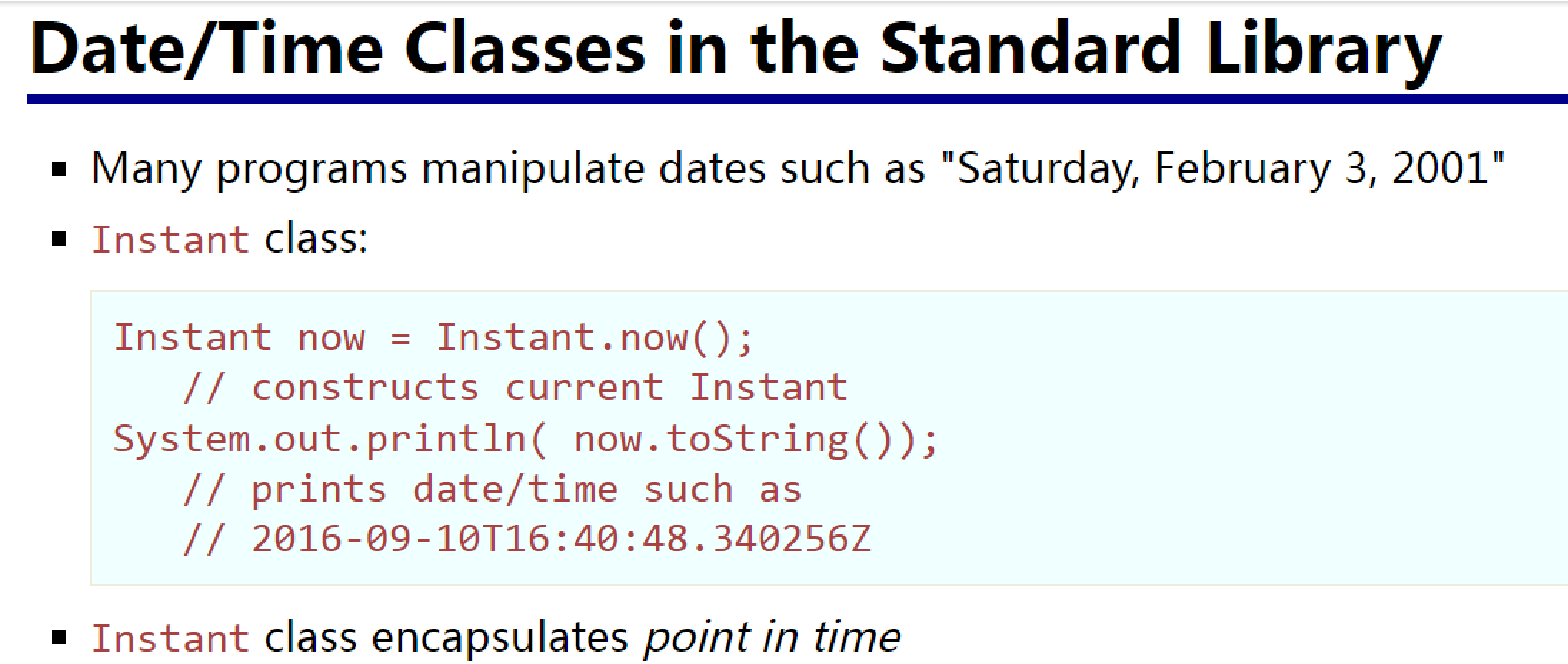
Lecture 2

- WarmUp: The date and time classes in Java APIs
- WarmUp: Time Measuring of Programs
- Introduction to Analysis of Algorithms (week 2,3)
  (1.4 of Text A; Ch3 of Text B)

To be discussed in Lecture 3,4:

- Basics of Algorithm Design Methods
  (1.1 of TextA; Ch1, Ch2 of Text B)
- Lists, Stacks & Queues (1.3 of Text A)

WarmUp: The date and time classes in Java APIs



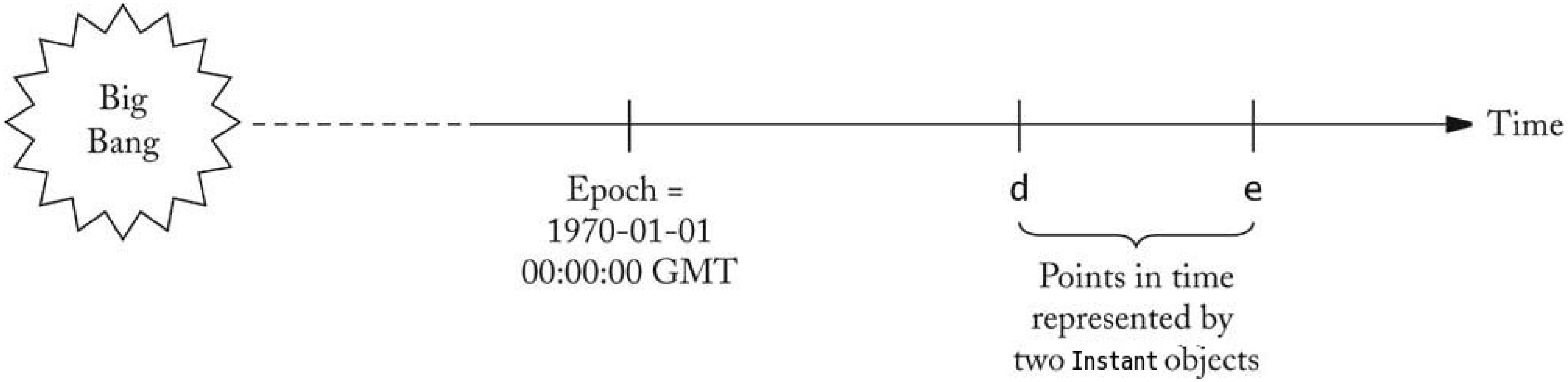
Date/Time Classes in the Standard Library

- Many programs manipulate dates such as "Saturday, February 3, 2001"
- `Instant` class:

```
Instant now = Instant.now();
   // constructs current Instant
System.out.println( now.toString());
   // prints date/time such as
   // 2016-09-10T16:40:48.340256Z
```

- `Instant` class encapsulates *point in time*

Points in Time


Big Bang
Time
Epoch = 1970-01-01 00:00:00 GMT
d
e
Points in time represented by two Instant objects

Methods of the `Instant` class

Method	Description
`long getEpochSecond()`	Gets the number of seconds since the epoch.
`long getNano()`	Gets the number of nanoseconds since the last second.

`Instant plusSeconds(long seconds)`
`Instant plusNanos(long nanos)`
	Yields the instant that is obtained by adding the given number of seconds or nanoseconds.

`Instant plus(Duration duration)`	Yields the instant that is obtained by adding the given `Duration` (which encapulates seconds and nanoseconds).
`ZonedDateTime atZone(ZoneId zone)`	Yields the `ZonedDateTime` at a given time zone. You get a `ZoneId` with its static `of` method, such as `ZoneId.of("America/Los_Angeles")`.
`String toString()`	Yields a representation in ISO-8601 format.

The `ZonedDateTime` Class

- The `Instant` class doesn't measure months, weekdays, etc.
- That's the job of a *calendar*
- A calendar assigns a name to a point in time
- Many calendars in use:
  - Gregorian
  - Contemporary: Hebrew, Arabic, Chinese
  - Historical: French Revolutionary, Mayan
- The `ZonedDateTime` class uses the Gregorian calendar, and it knows about time zones.
- Legacy classes `Date`, `GregorianCalendar`

Methods of the `ZonedDateTime` Class

Method	Description

`int getYear()`
`int getMonthValue()`
`int getDayOfMonth()`
	Gets the year, month, or day.

`DayOfWeek getDayOfWeek()`	Gets the day of the week. Call the `value` method on the returned object to get an integer value (1 = Monday … 7 = Sunday).

`int getHour()`
`int getMinute()`
`int getSecond()`
`int getNano()`
	Gets the hour, minute, second, or nanosecond of this `ZonedDateTime`.

`ZoneOffset getOffset()`	Gets the offset from the zero meridian. Call `getTotalSeconds` on the returned object to get the offset in seconds.

`ZonedDateTime plusDays(int n)`
`ZonedDateTime plusWeeks(int n)`
`ZonedDateTime plusMonths(int n)`
`ZonedDateTime plusYears(int n)`
`ZonedDateTime plusHours(int n)`
`ZonedDateTime plusMinutes(int n)`
`ZonedDateTime plusSeconds(int n)`
`ZonedDateTime plusNanos(int n)`
	Yields a `ZonedDateTime` that is obtained by adding a the given number of days, weeks, months, years, hours, minutes, seconds, or nanoseconds temporal units to this `ZonedDateTime`.

WarmUp: Measuring Execution Time of Programs

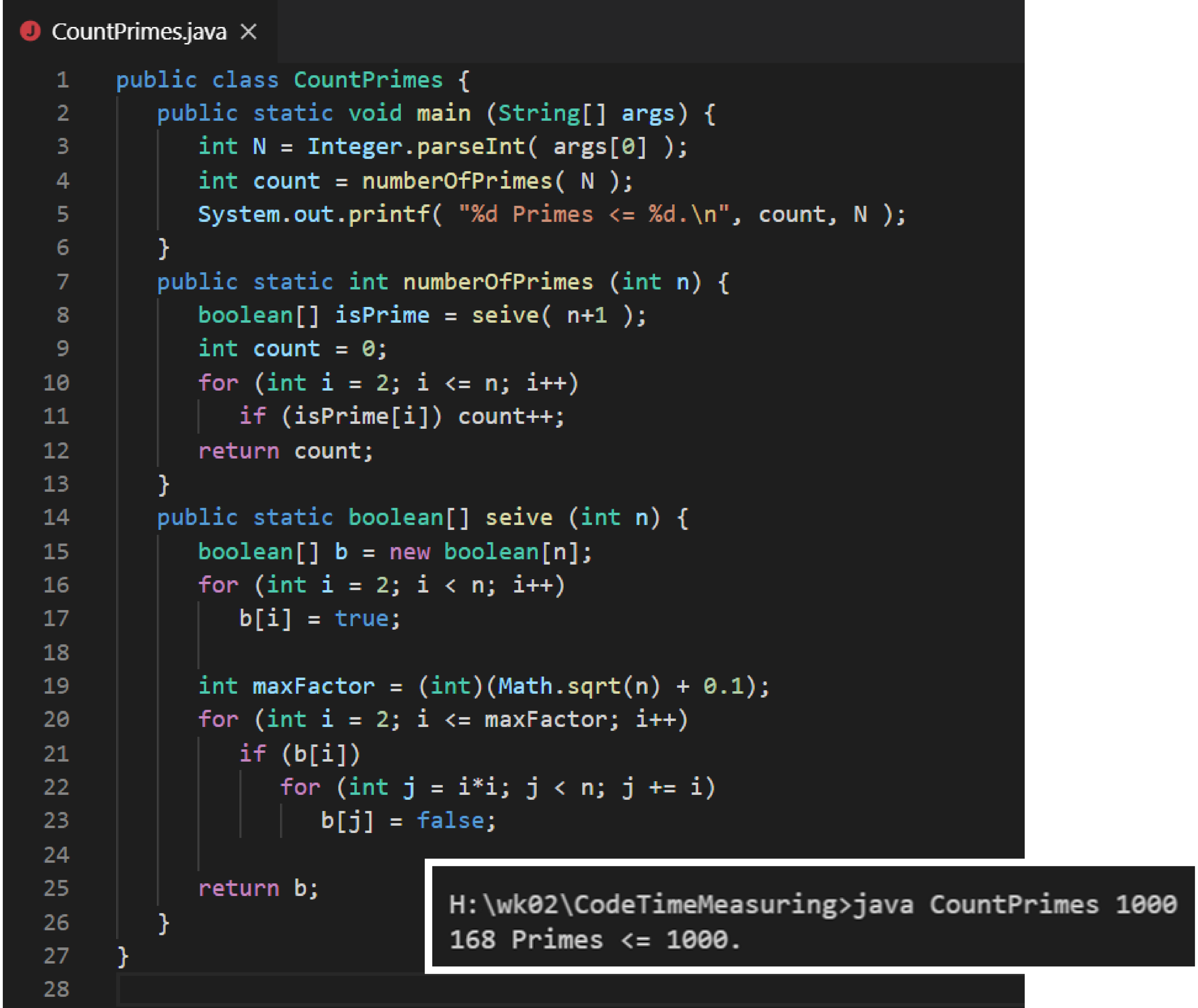
Explore the time consumption of CountPrimes and PrimeCounter given below with the following test frameworks (in Java):

```
import java.util.Date;
Date start = new Date();   // JDK 1.0+
…  // Do something by tested task
Date end = new Date();
long timeInMS = end.getTime() – start.getTime();
```

Or

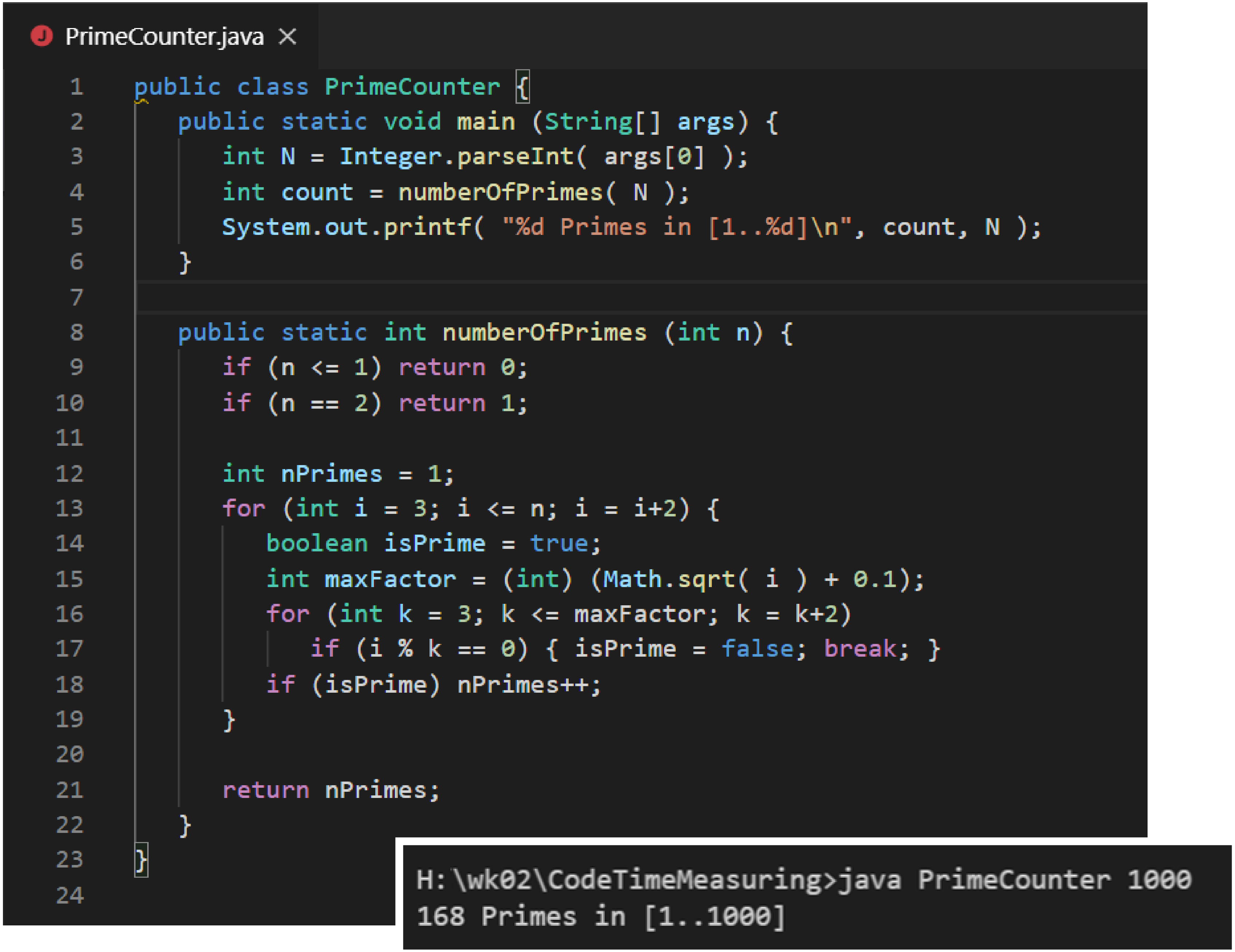
```
import java.time.Instant;
import java.time.Duration;
Instant start = Instant.now();   // JDK 8.0+
…   // Do something by tested task
Instant end = Instant.now();
long timeInMS = Duration.between(start, end).toMillis();
```

Alternatives in C/C++ : <time.h> for C programs.



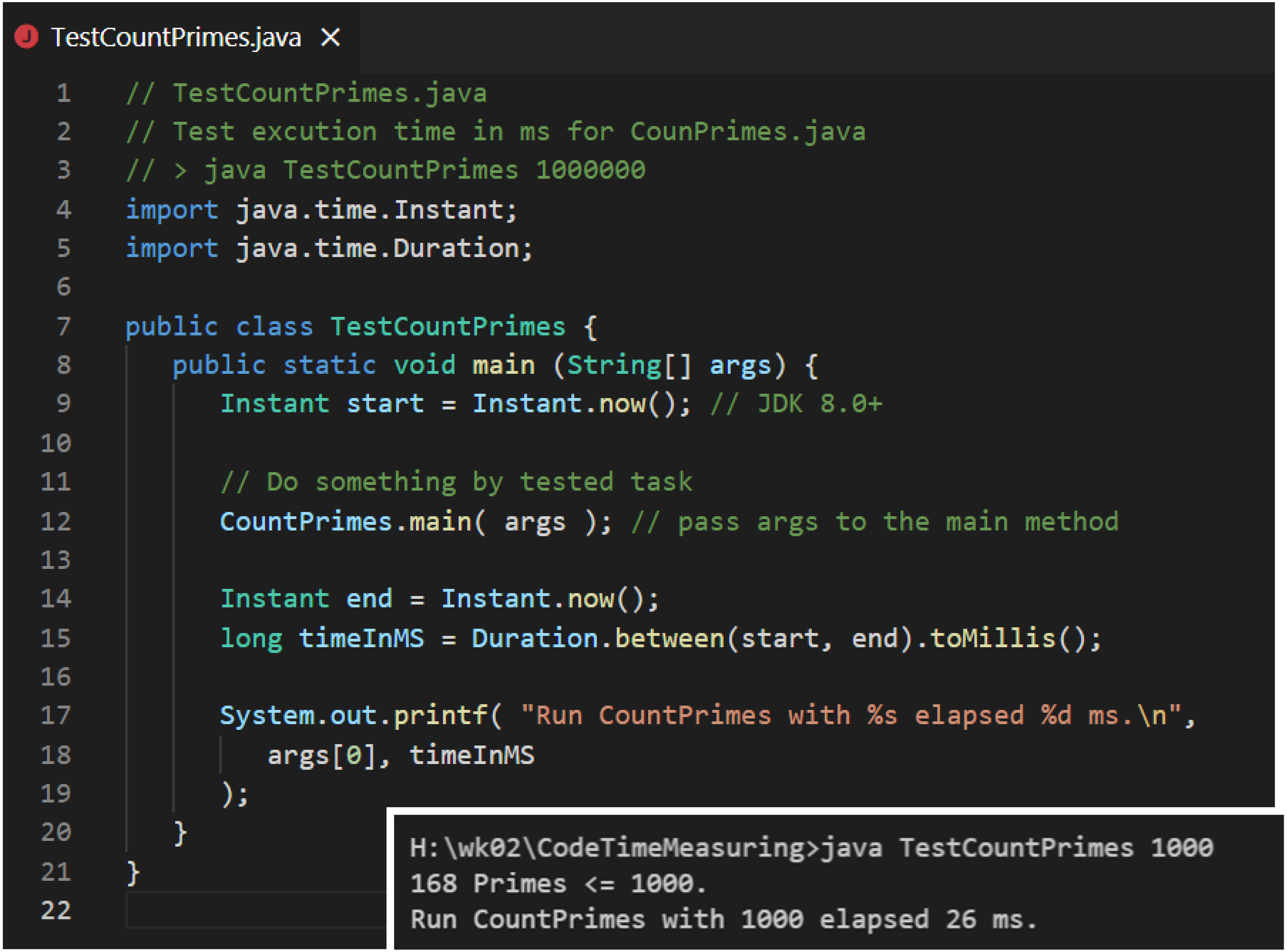
```java
public class CountPrimes {
    public static void main (String[] args) {
        int N = Integer.parseInt( args[0] );
        int count = numberOfPrimes( N );
        System.out.printf( "%d Primes <= %d.\n", count, N );
    }
    public static int numberOfPrimes (int n) {
        boolean[] isPrime = seive( n+1 );
        int count = 0;
        for (int i = 2; i <= n; i++)
            if (isPrime[i]) count++;
        return count;
    }
    public static boolean[] seive (int n) {
        boolean[] b = new boolean[n];
        for (int i = 2; i < n; i++)
            b[i] = true;

        int maxFactor = (int)(Math.sqrt(n) + 0.1);
        for (int i = 2; i <= maxFactor; i++)
            if (b[i])
                for (int j = i*i; j < n; j += i)
                    b[j] = false;

        return b;
    }
}
```

```
H:\wk02\CodeTimeMeasuring>java CountPrimes 1000
168 Primes <= 1000.
```



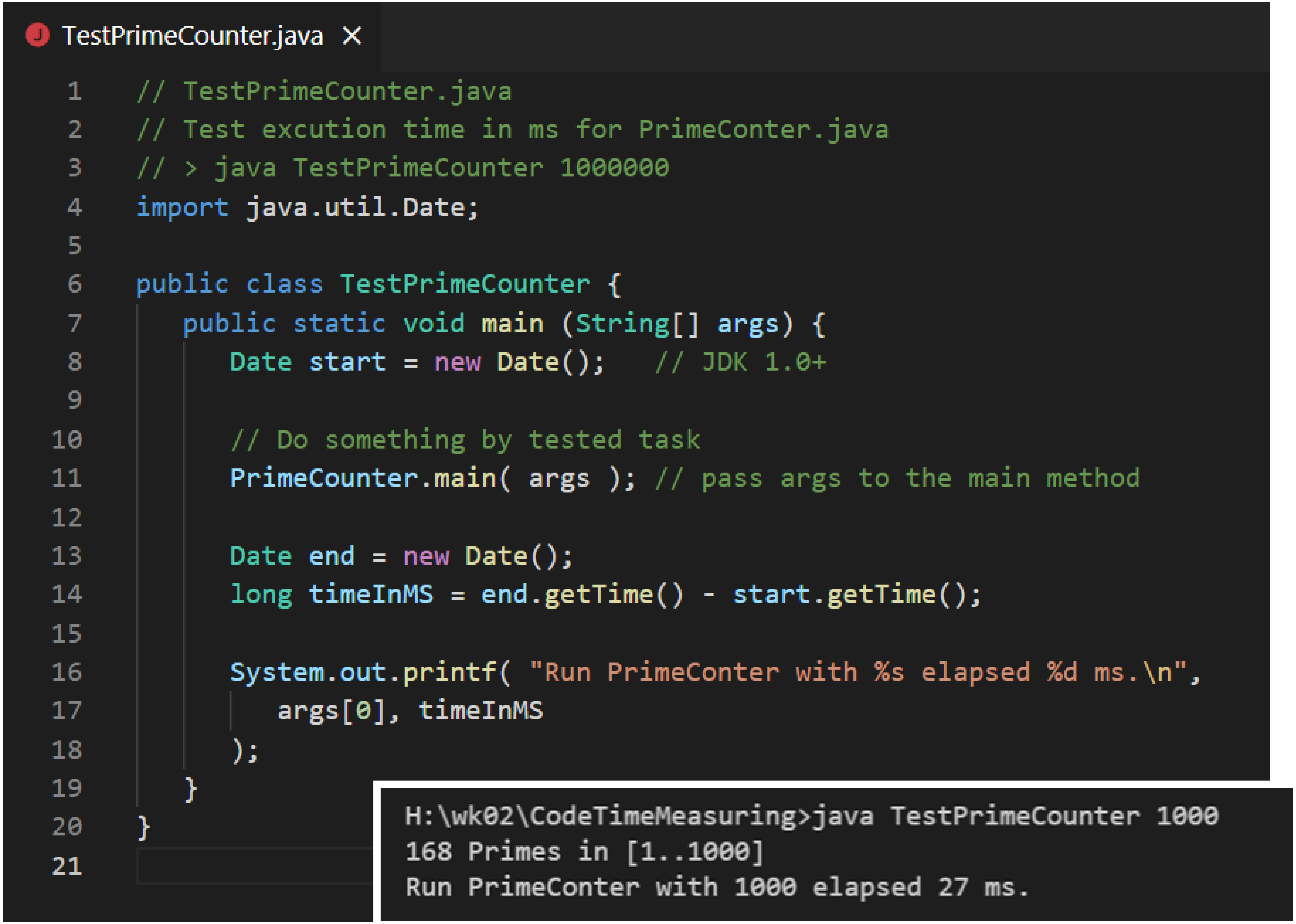
```java
public class PrimeCounter {
    public static void main (String[] args) {
        int N = Integer.parseInt( args[0] );
        int count = numberOfPrimes( N );
        System.out.printf( "%d Primes in [1..%d]\n", count, N );
    }

    public static int numberOfPrimes (int n) {
        if (n <= 1) return 0;
        if (n == 2) return 1;

        int nPrimes = 1;
        for (int i = 3; i <= n; i = i+2) {
            boolean isPrime = true;
            int maxFactor = (int) (Math.sqrt( i ) + 0.1);
            for (int k = 3; k <= maxFactor; k = k+2)
                if (i % k == 0) { isPrime = false; break; }
            if (isPrime) nPrimes++;
        }

        return nPrimes;
    }
}

```

```
H:\wk02\CodeTimeMeasuring>java PrimeCounter 1000
168 Primes in [1..1000]
```



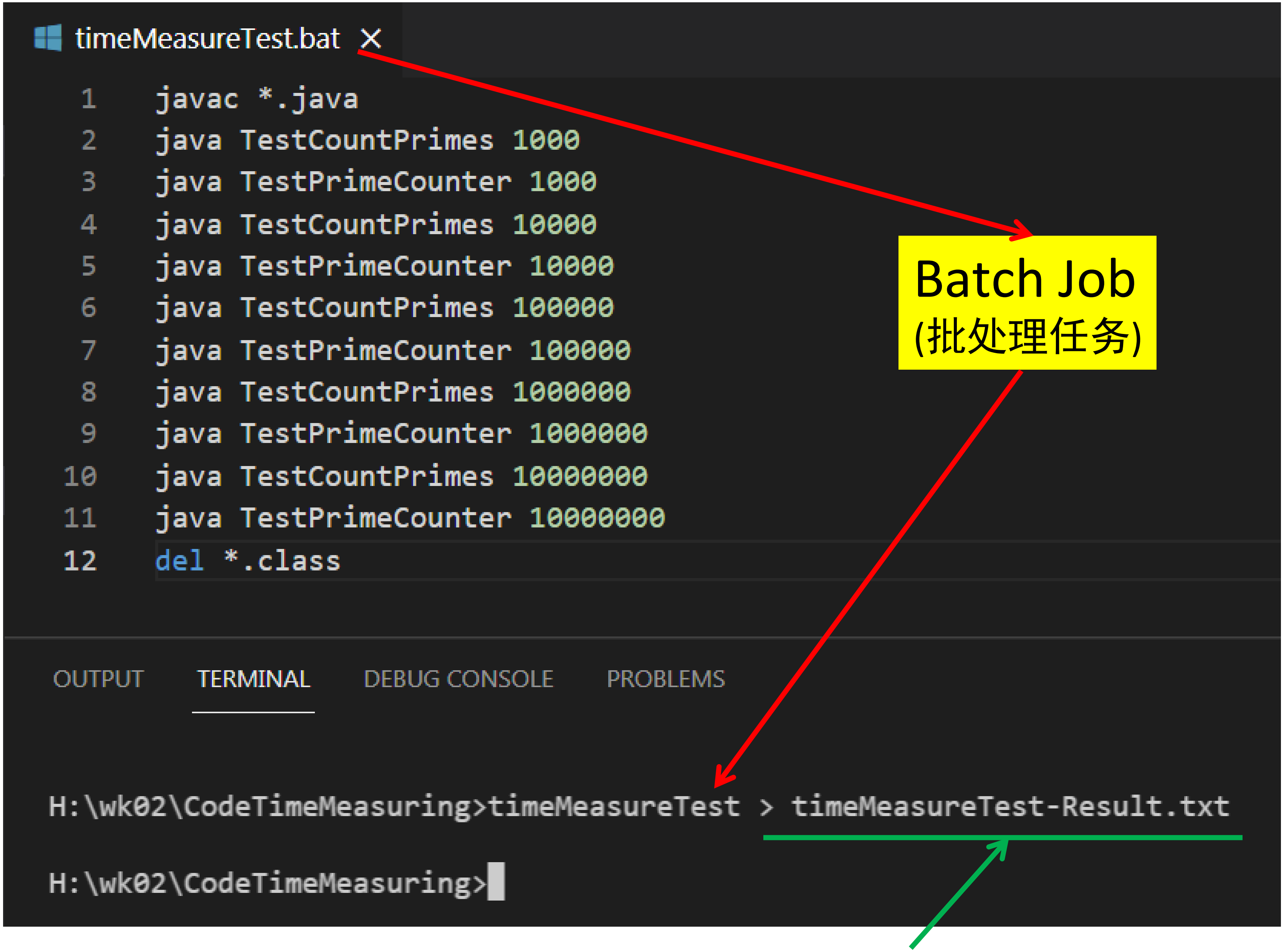
```java
// TestCountPrimes.java
// Test excution time in ms for CounPrimes.java
// > java TestCountPrimes 1000000
import java.time.Instant;
import java.time.Duration;

public class TestCountPrimes {
    public static void main (String[] args) {
        Instant start = Instant.now(); // JDK 8.0+

        // Do something by tested task
        CountPrimes.main( args ); // pass args to the main method

        Instant end = Instant.now();
        long timeInMS = Duration.between(start, end).toMillis();

        System.out.printf( "Run CountPrimes with %s elapsed %d ms.\n",
            args[0], timeInMS
        );
    }
}

```

```
H:\wk02\CodeTimeMeasuring>java TestCountPrimes 1000
168 Primes <= 1000.
Run CountPrimes with 1000 elapsed 26 ms.
```



```java
// TestPrimeCounter.java
// Test excution time in ms for PrimeConter.java
// > java TestPrimeCounter 1000000
import java.util.Date;

public class TestPrimeCounter {
    public static void main (String[] args) {
        Date start = new Date();   // JDK 1.0+

        // Do something by tested task
        PrimeCounter.main( args ); // pass args to the main method

        Date end = new Date();
        long timeInMS = end.getTime() - start.getTime();

        System.out.printf( "Run PrimeConter with %s elapsed %d ms.\n",
            args[0], timeInMS
        );
    }
}

```

```
H:\wk02\CodeTimeMeasuring>java TestPrimeCounter 1000
168 Primes in [1..1000]
Run PrimeConter with 1000 elapsed 27 ms.
```



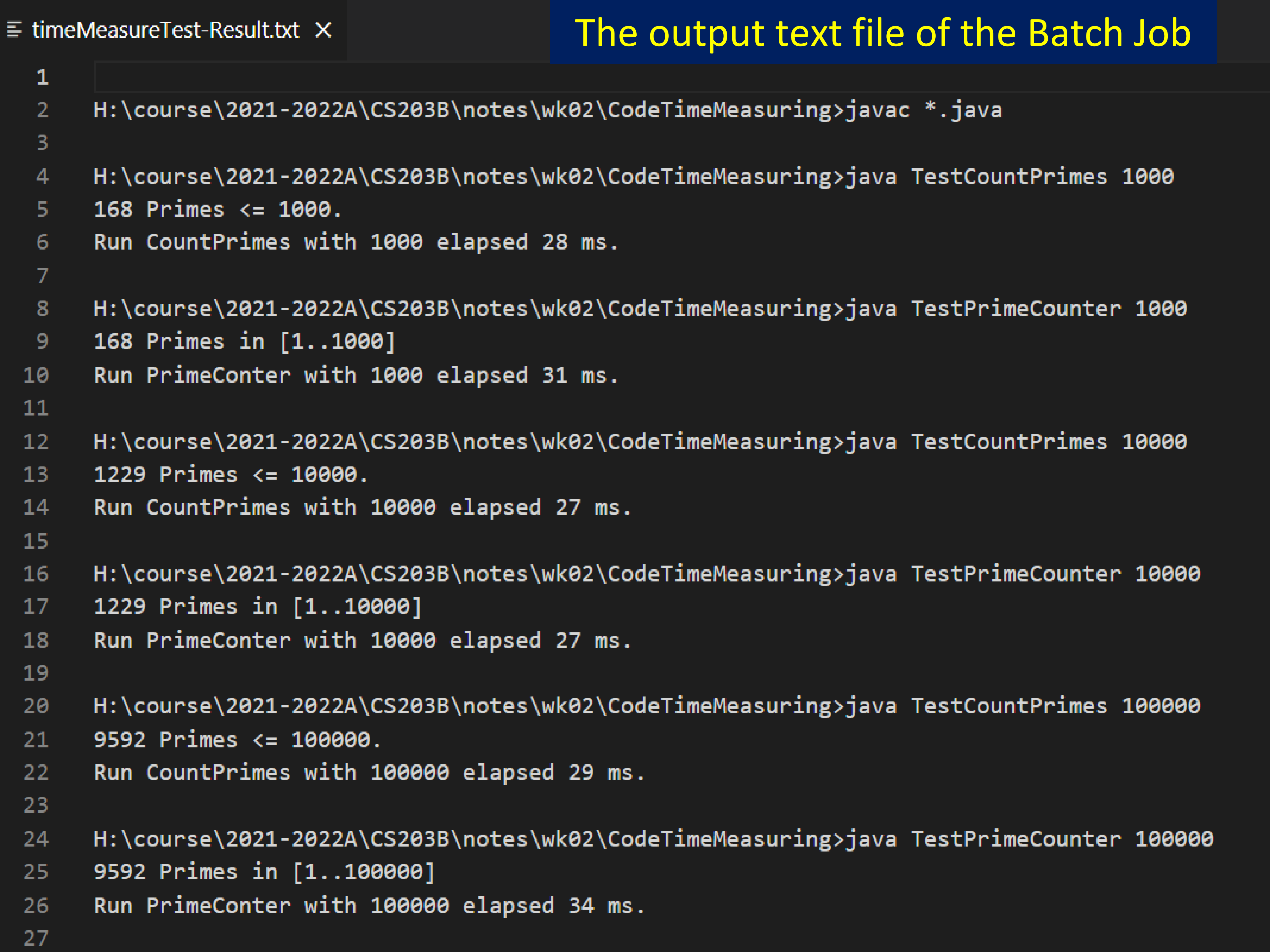
```
javac *.java
java TestCountPrimes 1000
java TestPrimeCounter 1000
java TestCountPrimes 10000
java TestPrimeCounter 10000
java TestCountPrimes 100000
java TestPrimeCounter 100000
java TestCountPrimes 1000000
java TestPrimeCounter 1000000
java TestCountPrimes 10000000
java TestPrimeCounter 10000000
del *.class
```

```
H:\wk02\CodeTimeMeasuring>timeMeasureTest > timeMeasureTest-Result.txt

H:\wk02\CodeTimeMeasuring>
```

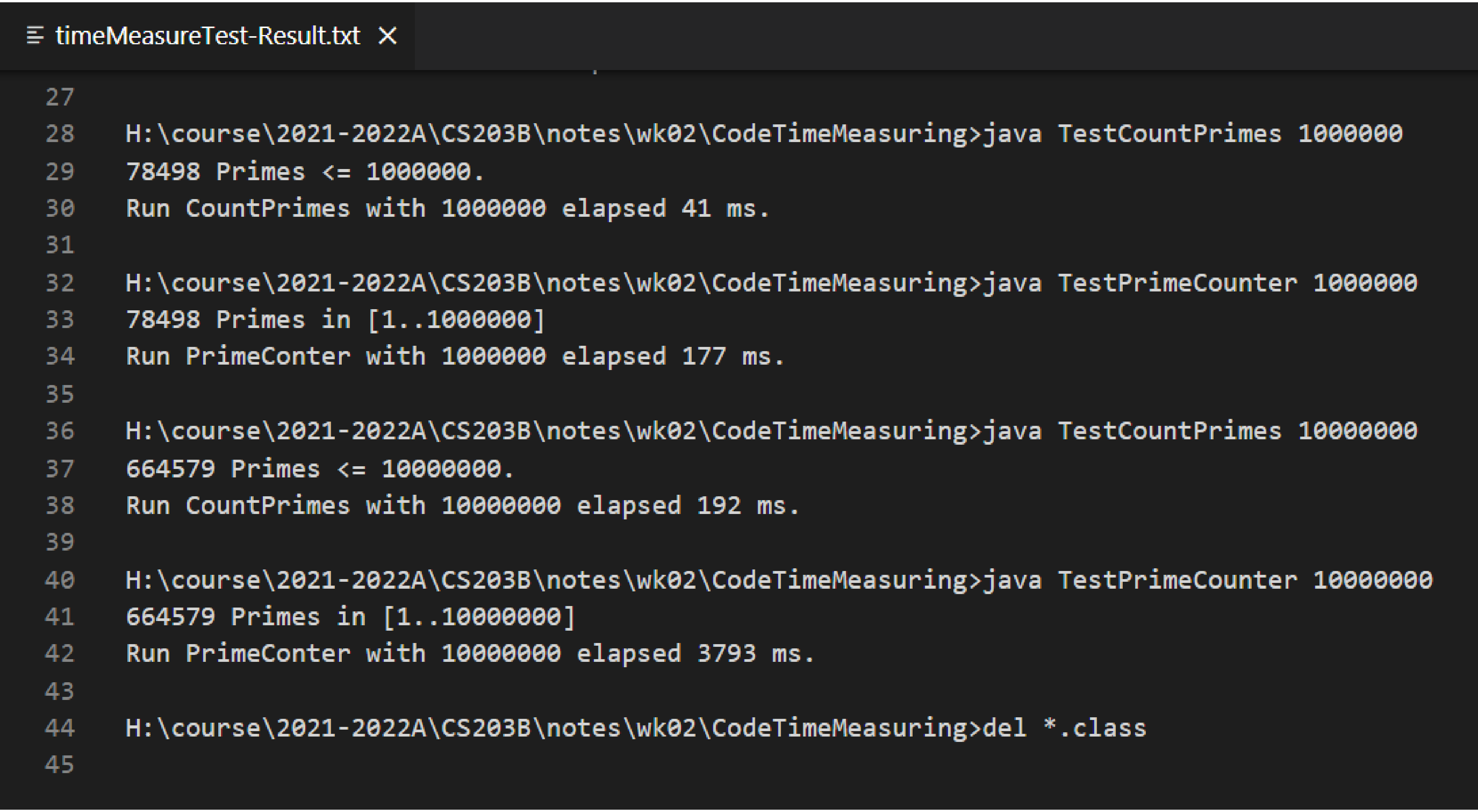
Batch Job
(批处理任务)

Output Redirection (输出重定向)

The output text file of the Batch Job



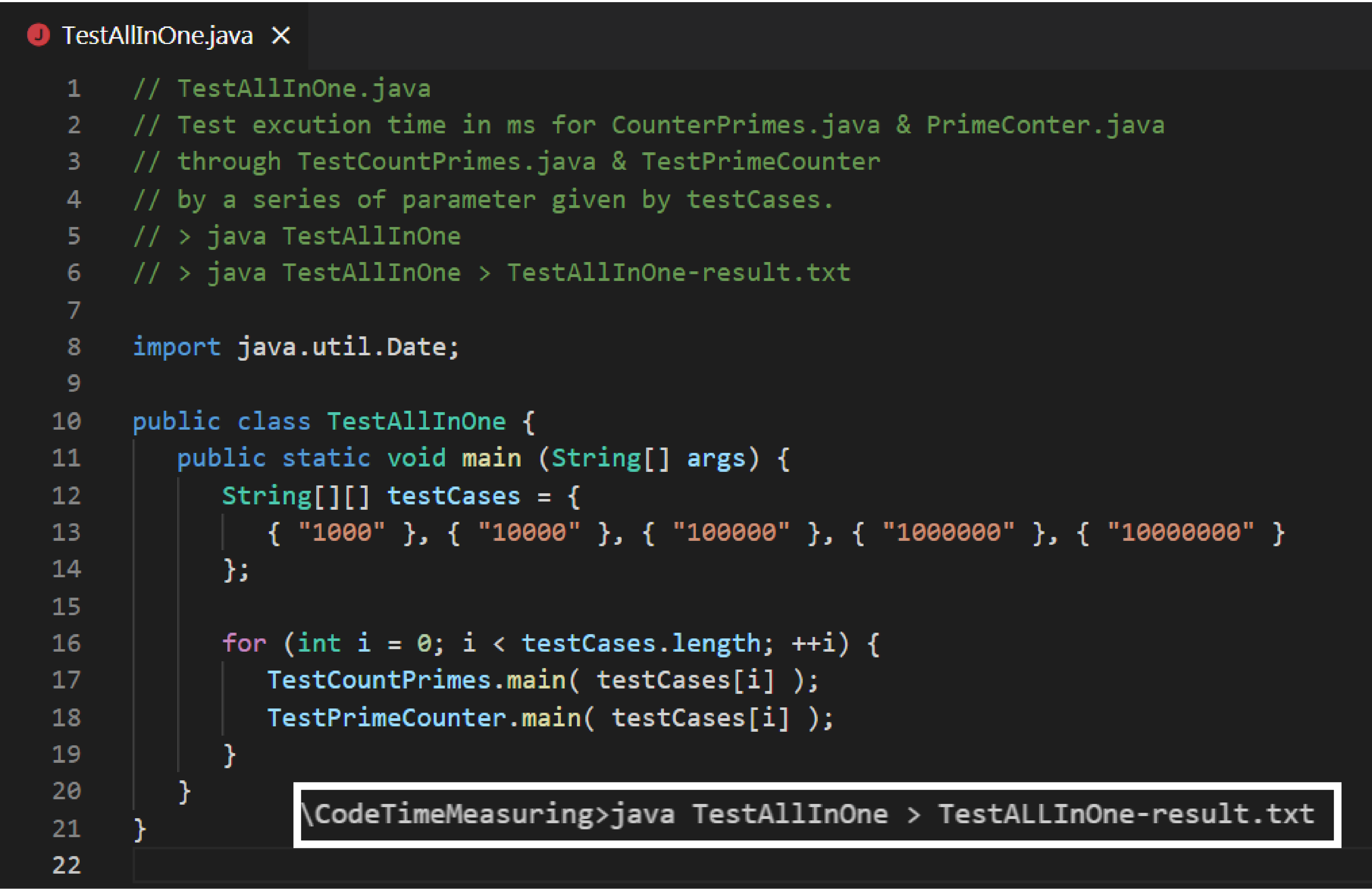
```

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>javac *.java

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestCountPrimes 1000
168 Primes <= 1000.
Run CountPrimes with 1000 elapsed 28 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestPrimeCounter 1000
168 Primes in [1..1000]
Run PrimeConter with 1000 elapsed 31 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestCountPrimes 10000
1229 Primes <= 10000.
Run CountPrimes with 10000 elapsed 27 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestPrimeCounter 10000
1229 Primes in [1..10000]
Run PrimeConter with 10000 elapsed 27 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestCountPrimes 100000
9592 Primes <= 100000.
Run CountPrimes with 100000 elapsed 29 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestPrimeCounter 100000
9592 Primes in [1..100000]
Run PrimeConter with 100000 elapsed 34 ms.

```



```

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestCountPrimes 1000000
78498 Primes <= 1000000.
Run CountPrimes with 1000000 elapsed 41 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestPrimeCounter 1000000
78498 Primes in [1..1000000]
Run PrimeConter with 1000000 elapsed 177 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestCountPrimes 10000000
664579 Primes <= 10000000.
Run CountPrimes with 10000000 elapsed 192 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>java TestPrimeCounter 10000000
664579 Primes in [1..10000000]
Run PrimeConter with 10000000 elapsed 3793 ms.

H:\course\2021-2022A\CS203B\notes\wk02\CodeTimeMeasuring>del *.class

```

Q: I don’t like to use batch job. Any Alternatives?

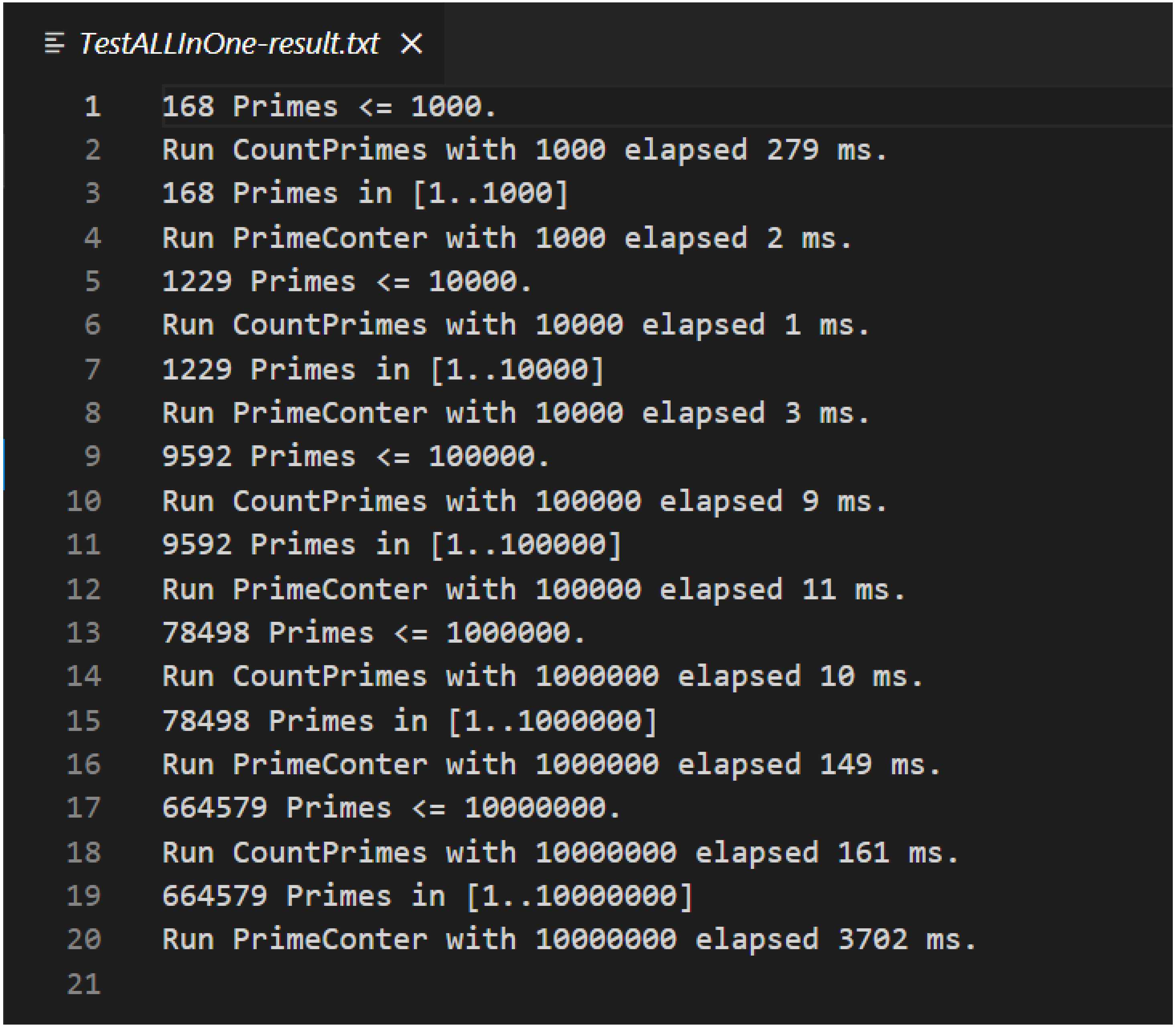
A: Yes. We can write loops in programs.



```java
// TestAllInOne.java
// Test excution time in ms for CounterPrimes.java & PrimeConter.java
// through TestCountPrimes.java & TestPrimeCounter
// by a series of parameter given by testCases.
// > java TestAllInOne
// > java TestAllInOne > TestAllInOne-result.txt

import java.util.Date;

public class TestAllInOne {
    public static void main (String[] args) {
        String[][] testCases = {
            { "1000" }, { "10000" }, { "100000" }, { "1000000" }, { "10000000" }
        };

        for (int i = 0; i < testCases.length; ++i) {
            TestCountPrimes.main( testCases[i] );
            TestPrimeCounter.main( testCases[i] );
        }
    }
}
```

```
\CodeTimeMeasuring>java TestAllInOne > TestALLInOne-result.txt
```

We get clean results: ☺



```
168 Primes <= 1000.
Run CountPrimes with 1000 elapsed 279 ms.
168 Primes in [1..1000]
Run PrimeConter with 1000 elapsed 2 ms.
1229 Primes <= 10000.
Run CountPrimes with 10000 elapsed 1 ms.
1229 Primes in [1..10000]
Run PrimeConter with 10000 elapsed 3 ms.
9592 Primes <= 100000.
Run CountPrimes with 100000 elapsed 9 ms.
9592 Primes in [1..100000]
Run PrimeConter with 100000 elapsed 11 ms.
78498 Primes <= 1000000.
Run CountPrimes with 1000000 elapsed 10 ms.
78498 Primes in [1..1000000]
Run PrimeConter with 1000000 elapsed 149 ms.
664579 Primes <= 10000000.
Run CountPrimes with 10000000 elapsed 161 ms.
664579 Primes in [1..10000000]
Run PrimeConter with 10000000 elapsed 3702 ms.
```

命令行操作 vs IDEs

- 基本操作/基本功
- 揭示背后的原理
- 以不变应万变
- 易学/长期可用
- 不变之道

- 诸多功能/专业化
- 封装底层操作
- 方便法门/效率
- 持续学习/更新
- 易变之术

Summary

- WarmUp: The date and time classes in Java APIs
- WarmUp: Time Measuring of Programs
- Introduction to Analysis of Algorithms (week 2,3)
  (1.4 of Text A; Ch3 of Text B)

To be discussed in Lecture 3,4:

- Basics of Algorithm Design Methods
  (1.1 of TextA; Ch1, Ch2 of Text B)
- Lists, Stacks & Queues (1.3 of Text A)