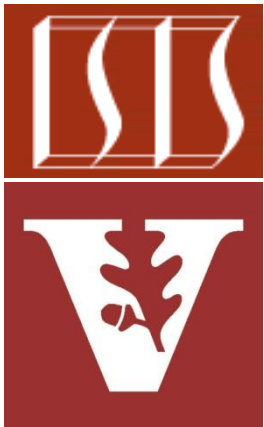


Java CyclicBarrier: Example Application



Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

**Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of Java CyclicBarrier
- Recognize the key methods in the Java CyclicBarrier
- Know how to program with Java CyclicBarrier in practice

```
class GCDCyclicBarrierWorker implements Runnable {
    private final CyclicBarrier mEntryBarrier;
    private final CyclicBarrier mExitBarrier; ...

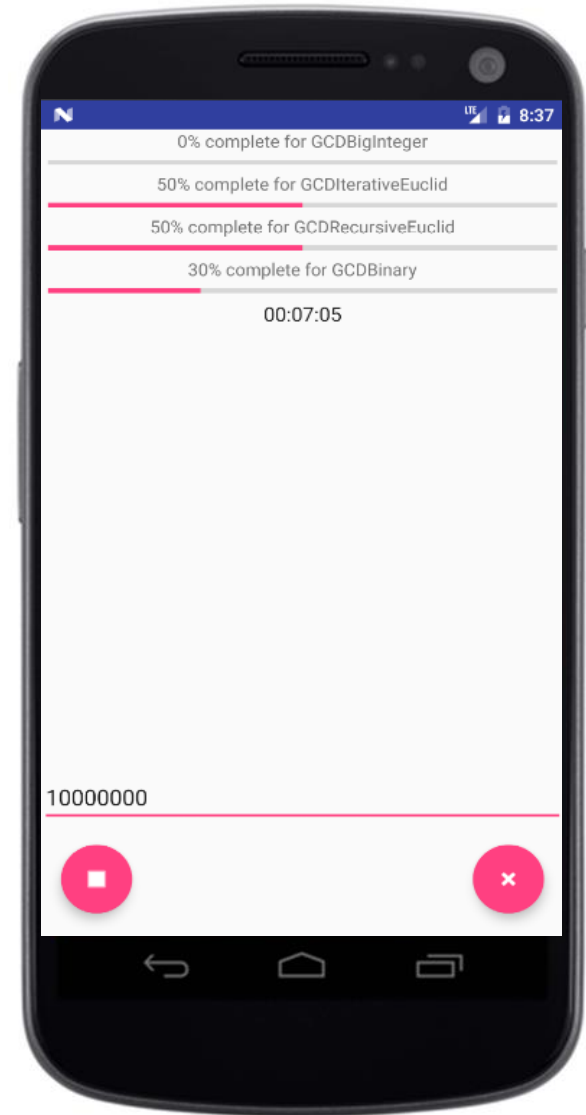
    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,
                           CyclicBarrier exitBarrier, ...) {
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;
        ...
    }

    public void run() {
        ...
        mEntryBarrier.await();
        runTest();
        mExitBarrier.await();
        ...
    }
}
```

Overview of the GCD App

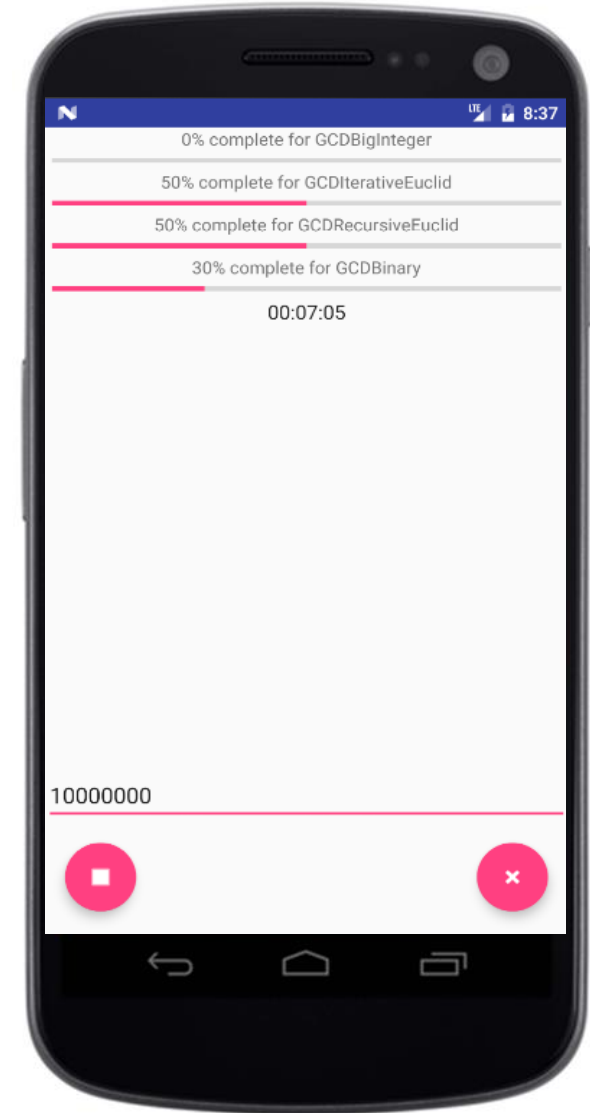
Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms



Overview of the GCD App

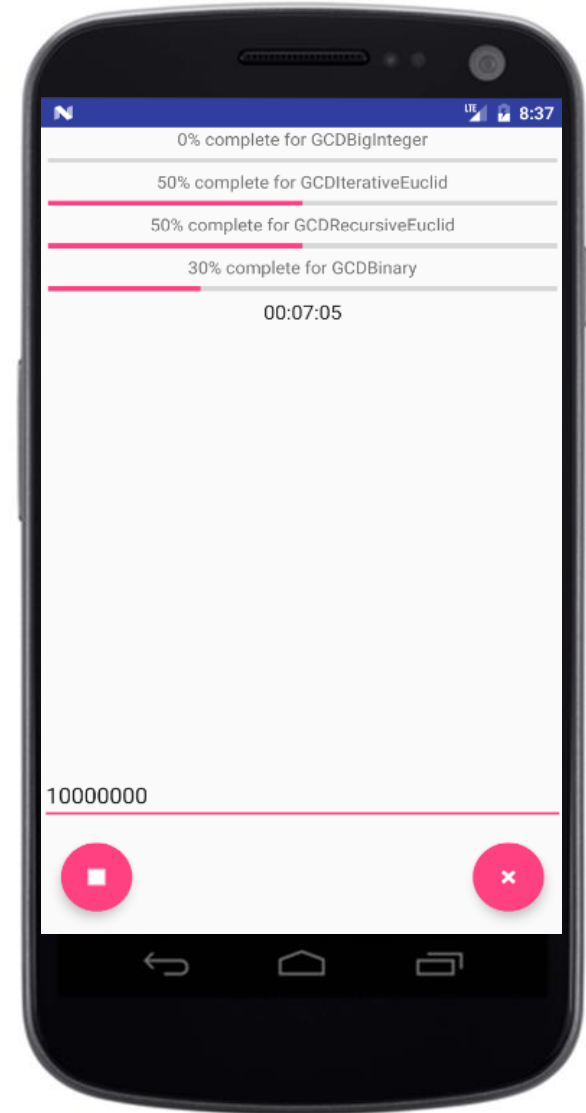
- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - e.g., the GCD of 80 & 120 = 40



See en.wikipedia.org/wiki/Greatest_common_divisor

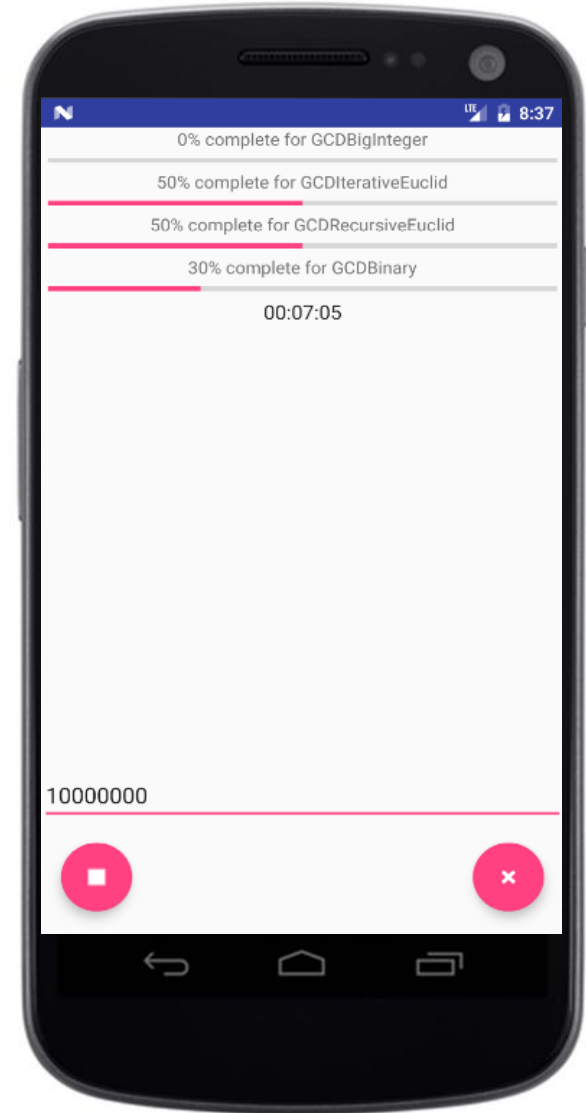
Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - Four GCD algorithms are tested



Overview of the GCD App

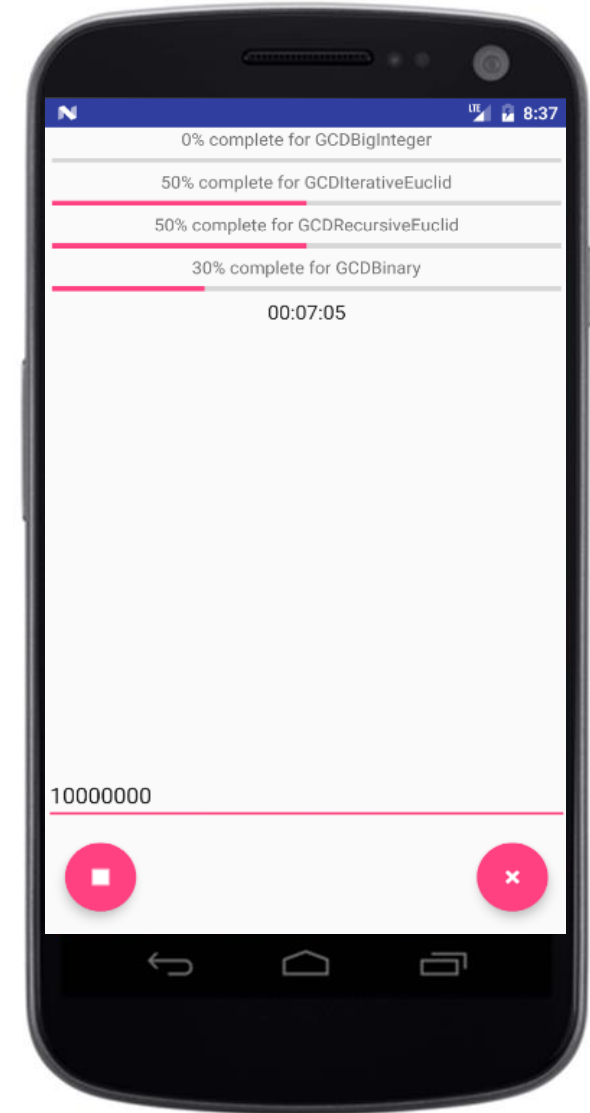
- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - Four GCD algorithms are tested
 - The gcd() method defined by BigInteger



See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html#gcd

Overview of the GCD App

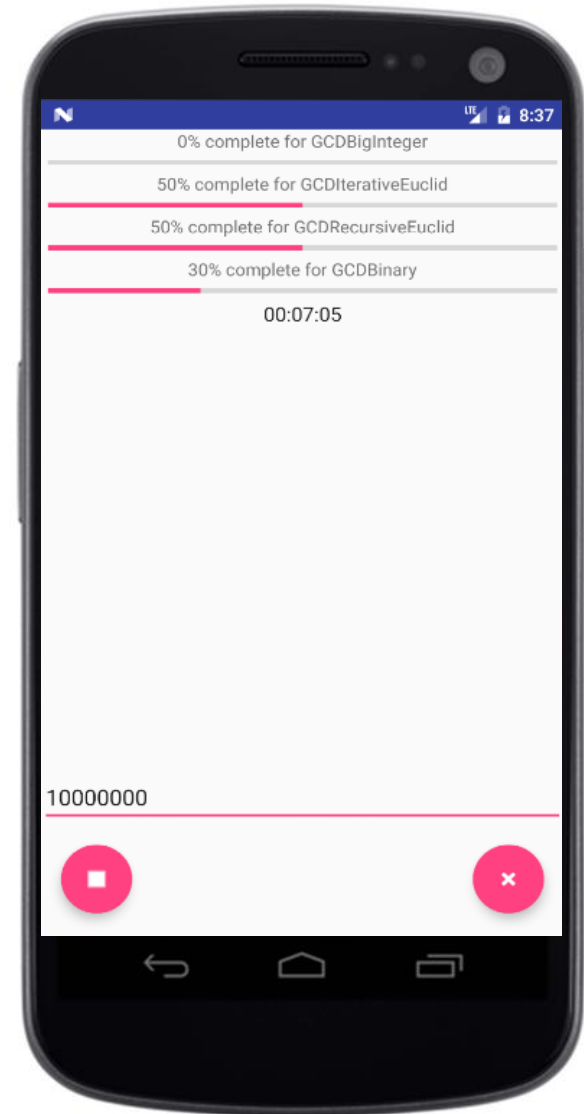
- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - Four GCD algorithms are tested
 - The gcd() method defined by BigInteger
 - An iterative Euclid algorithm



See en.wikipedia.org/wiki/Euclidean_algorithm

Overview of the GCD App

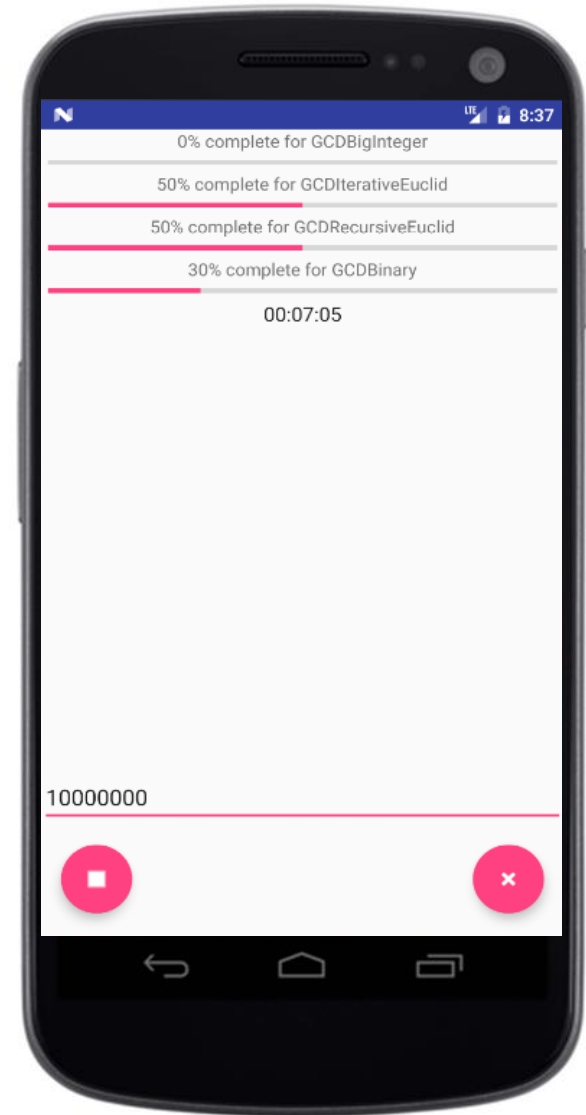
- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - Four GCD algorithms are tested
 - The gcd() method defined by BigInteger
 - An iterative Euclid algorithm
 - A recursive Euclid algorithm



See codedost.com/java/methods-and-recursion-in-java/java-program-to-find-gcd-hcf-using-euclidean-algorithm-using-recursion

Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - Four GCD algorithms are tested
 - The gcd() method defined by BigInteger
 - An iterative Euclid algorithm
 - A recursive Euclid algorithm
 - A complex GCD algorithm that uses binary arithmetic

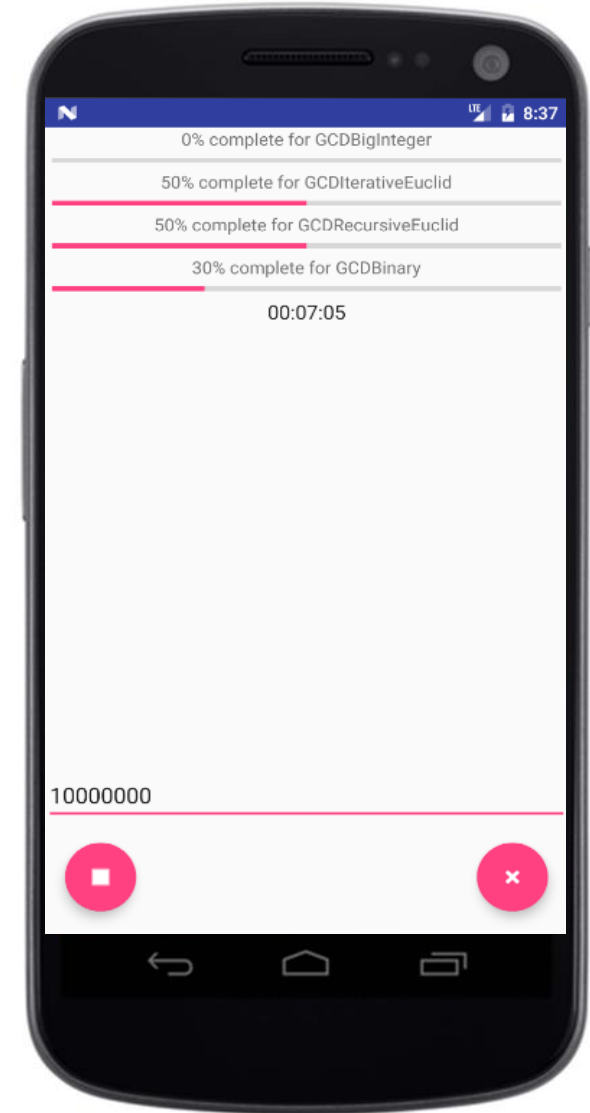


See en.wikipedia.org/wiki/Binary_GCD_algorithm

Overview of the GCD App

- This Android app uses CyclicBarrier objects to coordinate the concurrent benchmarking of four Greatest Common Divisor (GCD) algorithms
 - GCD computes the largest positive integer that is a divisor of two numbers
 - Four GCD algorithms are tested
 - The gcd() method defined by BigInteger
 - An iterative Euclid
 - A recursive Euclid
 - A combinatorial

NOT
Important



However, the details of these algorithms are not important for our discussion

GCDCyclicBarrierTest Class Walkthrough

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```

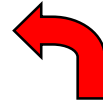
See [GCD/CyclicBarrier/app/src/test/java/edu/vandy/gcdtesttask/GCDCyclicBarrierTest.java](https://github.com/vandy/gcdtesttask/GCDCyclicBarrierTest.java)

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {  
    @Test public void testGCDCyclicBarrierTest() {  
        List<GCDTuple> gcdTests = makeGCDTuples();  
  
        CyclicBarrier entryBarrier =  
            new CyclicBarrier(gcdTests.size() + 1, () ->  
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));  
        CyclicBarrier exitBarrier =  
            new CyclicBarrier(gcdTests.size() + 1);  
  
        for (int cycle = 1; cycle <= sCYCLES; cycle++) {  
            gcdTests.forEach(gcdTuple -> new Thread(new  
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,  
                    gcdTuple, this)).start());  
            System.out.println("Starting tests");  
            entryBarrier.await();  
            System.out.println("Waiting for results");  
            exitBarrier.await();  
            System.out.println("All tests done"); ...  
        }  
    }  
}
```

Entry point into
the unit test

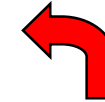


GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {  
    @Test public void testGCDCyclicBarrierTester() {  
        List<GCDTuple> gcdTests = makeGCDTuples();  
  
        CyclicBarrier entryBarrier =  
            new CyclicBarrier(gcdTests.size() + 1, () ->  
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));  
        CyclicBarrier exitBarrier =  
            new CyclicBarrier(gcdTests.size() + 1);  
  
        for (int cycle = 1; cycle <= sCYCLES; cycle++) {  
            gcdTests.forEach(gcdTuple -> new Thread(new  
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,  
                    gcdTuple, this)).start());  
            System.out.println("Starting tests");  
            entryBarrier.await();  
            System.out.println("Waiting for results");  
            exitBarrier.await();  
            System.out.println("All tests done"); ...  
        }  
    }  
}
```

Initialize all the GCD algorithms



GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
```

```
    @Test public void testGCDCyclicBarrierTester() {
```

```
        List<GCDTuple> gcdTests = makeGCDTuples();
```

```
        CyclicBarrier entryBarrier =  Create entry barrier
```

```
            new CyclicBarrier(gcdTests.size() + 1, () ->
```

```
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
```

```
        CyclicBarrier exitBarrier =
```

```
            new CyclicBarrier(gcdTests.size() + 1);
```

```
        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
```

```
            gcdTests.forEach(gcdTuple -> new Thread(new
```

```
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,  
                    gcdTuple, this)).start());
```

```
            System.out.println("Starting tests");
```

```
            entryBarrier.await();
```

```
            System.out.println("Waiting for results");
```

```
            exitBarrier.await();
```

```
            System.out.println("All tests done"); ...
```

We add a "+ 1" for the thread that initializes the tests

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
```

```
    @Test public void testGCDCyclicBarrierTester() {
```

```
        List<GCDTuple> gcdTests = makeGCDTuples();
```

Barrier action allocates each cycle's input

```
        CyclicBarrier entryBarrier =
```

```
            new CyclicBarrier(gcdTests.size() + 1, () ->
```

```
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
```

```
        CyclicBarrier exitBarrier =
```

```
            new CyclicBarrier(gcdTests.size() + 1);
```

```
        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
```

```
            gcdTests.forEach(gcdTuple -> new Thread(new
```

```
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,  
                    gcdTuple, this)).start());
```

```
            System.out.println("Starting tests");
```

```
            entryBarrier.await();
```

```
            System.out.println("Waiting for results");
```

```
            exitBarrier.await();
```

```
            System.out.println("All tests done"); ...
```

GCD cyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDcyclicBarrierTest {
    @Test public void testGCDcyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDcyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =  Create exit barrier
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDcyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```

We add a "+ 1" for the thread that initializes the tests

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);
        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```

Iterate through
each cycle



GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```



Create & start
threads w/barriers

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());
            System.out.println("Starting tests");  Don't start just yet
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();
            System.out.println("All tests done"); ...
        }
    }
}
```

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());

            System.out.println("Starting tests");
            entryBarrier.await();  Let all worker threads proceed at the
            System.out.println("Waiting for results"); same time, fixing
            exitBarrier.await(); limitation with CountdownLatch
            System.out.println("All tests done"); ...
        }
    }
}
```

See previous lesson on "*Java CountdownLatch*"

GCDCyclicBarrierTest Class Walkthrough

- Create worker threads that use exit & entry barrier CyclicBarrier objects

```
class GCDCyclicBarrierTest {
    @Test public void testGCDCyclicBarrierTester() {
        List<GCDTuple> gcdTests = makeGCDTuples();

        CyclicBarrier entryBarrier =
            new CyclicBarrier(gcdTests.size() + 1, () ->
                GCDCyclicBarrierWorker.initializeInput(sITERATIONS));
        CyclicBarrier exitBarrier =
            new CyclicBarrier(gcdTests.size() + 1);

        for (int cycle = 1; cycle <= sCYCLES; cycle++) {
            gcdTests.forEach(gcdTuple -> new Thread(new
                GCDCyclicBarrierWorker(entryBarrier, exitBarrier,
                    gcdTuple, this)).start());
            System.out.println("Starting tests");
            entryBarrier.await();
            System.out.println("Waiting for results");
            exitBarrier.await();  Exit barrier waits for all threads to
            System.out.println("All tests done"); ... finish this cycle
        }
    }
}
```

After await() returns for a CyclicBarrier it will be reset (& is thus reusable) *without* needing to create a new CyclicBarrier instance

GCDCyclicBarrierWorker Class Walkthrough

GCD cyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```
class GCDcyclicBarrierWorker implements Runnable {  
    private final CyclicBarrier mEntryBarrier;  
    private final CyclicBarrier mExitBarrier;  
    ...  
  
    GCDcyclicBarrierWorker(CyclicBarrier entryBarrier,  
                           CyclicBarrier exitBarrier, ...) {  
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;  
        ...  
    }  
  
    public void run() {  
        ...  
        mEntryBarrier.await();  
        runTest();  
        mExitBarrier.await();  
        ...  
    }  
}
```



**Define a worker
that runs in a thread**

See [GCD/CyclicBarrier/app/src/main/java/edu/vandy/
gcdtesttask/presenter/GCDcyclicBarrierWorker.java](https://github.com/vandymc/gcdtesttask/presenter/GCDcyclicBarrierWorker.java)

GCDCyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```
class GCDCyclicBarrierWorker implements Runnable {  
    private final CyclicBarrier mEntryBarrier;  
    private final CyclicBarrier mExitBarrier;  
    ...  
  
    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,  
                           CyclicBarrier exitBarrier, ...) {  
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;  
        ...  
    }  
  
    public void run() {  
        ...  
        mEntryBarrier.await();  
        runTest();  
        mExitBarrier.await();  
        ...  
    }  
}
```



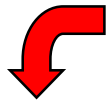
Initialize barrier fields

GCD cyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```
class GCDcyclicBarrierWorker implements Runnable {  
    private final CyclicBarrier mEntryBarrier;  
    private final CyclicBarrier mExitBarrier;  
    ...
```

```
    GCDcyclicBarrierWorker(CyclicBarrier entryBarrier,  
                           CyclicBarrier exitBarrier, ...) {  
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;  
        ...  
    }
```



**This hook method executes
after the thread is started**

```
    public void run() {  
        ...  
        mEntryBarrier.await();  
        runTest();  
        mExitBarrier.await();  
        ...
```

GCD CyclicBarrierWorker Class Walkthrough

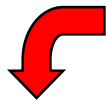
- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```
class GCD CyclicBarrierWorker implements Runnable {  
    private final CyclicBarrier mEntryBarrier;  
    private final CyclicBarrier mExitBarrier;  
    ...
```

```
    GCD CyclicBarrierWorker(CyclicBarrier entryBarrier,  
                             CyclicBarrier exitBarrier, ...) {  
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;  
        ...  
    }
```

```
    public void run() {  
        ...  
        mEntryBarrier.await();  
        runTest();  
        mExitBarrier.await();  
        ...
```

This entry barrier causes all worker threads to wait until they are all ready, thus fixing the earlier limitation with CountdownLatch



See previous lesson on "*Java CountdownLatch*"

GCD CyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```
class GCD CyclicBarrierWorker implements Runnable {  
    private final CyclicBarrier mEntryBarrier;  
    private final CyclicBarrier mExitBarrier;  
    ...
```

```
    GCD CyclicBarrierWorker(CyclicBarrier entryBarrier,  
                             CyclicBarrier exitBarrier, ...) {  
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;  
        ...  
    }
```

```
    public void run() {
```

```
        ...
```

```
        mEntryBarrier.await();
```

```
        runTest(); ← Run the GCD algorithm associated with this object
```

```
        mExitBarrier.await();
```

```
        ...
```



GCDCyclicBarrierWorker Class Walkthrough

- This class applies two entry & exit barrier CyclicBarrier objects to coordinate the benchmarking of a given GCD algorithm implementation

```
class GCDCyclicBarrierWorker implements Runnable {  
    private final CyclicBarrier mEntryBarrier;  
    private final CyclicBarrier mExitBarrier;  
    ...
```

```
    GCDCyclicBarrierWorker(CyclicBarrier entryBarrier,  
                           CyclicBarrier exitBarrier, ...) {  
        mEntryBarrier = entryBarrier; mExitBarrier = exitBarrier;  
        ...  
    }
```

```
    public void run() {  
        ...  
        mEntryBarrier.await();  
        runTest();  
        mExitBarrier.await();  
        ...
```



Exit barrier waits until all threads
are done before returning

End of Java CyclicBarrier: Example Application