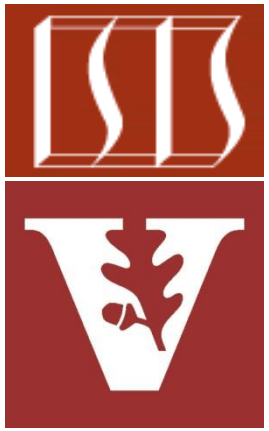
Example Application of Java Phaser



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 the Java Phaser barrier synchronizer
- Recognize the key methods in the Java Phaser
- Know how to program with Java Phaser in practice

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();
                new Thread(() -> {
                           entryPhaser.arriveAndAwaitAdvance();
                           task.run();
                           exitPhaser.arrive();
                          }).start();
    });
  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

This program expands on the pithy examples in the Java documentation at docs.oracle.com/javase/8/docs/api/java/util/concurrent/Phaser.html

Learning Objectives in this Part of the Lesson

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

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();
                new Thread(() -> {
                           entryPhaser.arriveAndAwaitAdvance();
                           task.run();
                           exitPhaser.arrive();
                          }).start();
    });
  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Showcases Phasers used as entry & exit barriers, in addition to one-shot & cyclic barriers

Test Driver Program Walkthrough

Test Driver Program Walkthrough

- Main entry point into the test program

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());



  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26

Test Driver Program Walkthrough

- Main entry point into the test program

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());



  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

A factory method that makes a list of MyTask objects

However, the details of what MyTask does are not important for our discussion

Test Driver Program Walkthrough

- Main entry point into the test program

Create a stream from 1 to sNUMBER_OF_TASKS

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());



  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

Test Driver Program Walkthrough

- Main entry point into the test program

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());



  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

Create a new MyTask object for each number in the stream

Test Driver Program Walkthrough

- Main entry point into the test program

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());



  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

Convert the stream into a list of MyTask objects

Test Driver Program Walkthrough

- Main entry point into the test program

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());


  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

Run a test showcasing one-shot Phasers used to run a list of tasks that all start at the same time

This method uses Phasers as an "entry barrier" & "exit barrier"

Test Driver Program Walkthrough

- Main entry point into the test program

```
private static List<MyTask> makeTasks() {
  return IntStream
    .rangeClosed(1, sNUMBER_OF_TASKS)

    .mapToObj(MyTask::new)

    .collect(toList());
}


static void main(String[] argv) {

  runOneShotTasks(makeTasks());

  runCyclicTasks(makeTasks(), sITERATIONS);
}
```

Run a test that showcases a cyclic Phaser that repeatedly performs actions for a given # of iterations

Applying a One-shot Phaser with Java

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();

                new Thread(() -> {
                           entryPhaser.arriveAndAwaitAdvance();

                           task.run();
                           exitPhaser.arrive();
                          }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                  entryPhaser.register();

                  new Thread(() -> {
                              entryPhaser.arriveAndAwaitAdvance();

                              task.run();
                              exitPhaser.arrive();
                            }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Entry barrier w/a "parties" value of 1 to register itself

This "entry barrier" Phaser is similar to a CyclicBarrier (but more flexible)

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                  entryPhaser.register();

                  new Thread(() -> {
                               entryPhaser.arriveAndAwaitAdvance();

                               task.run();
                               exitPhaser.arrive();
                             }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Exit barrier w/a "parties" value for all the tasks

This "exit barrier" Phaser is similar to a CountDownLatch (but more flexible)

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                  entryPhaser.register();

                  new Thread(() -> {
                               entryPhaser.arriveAndAwaitAdvance();

                               task.run();
                               exitPhaser.arrive();
                             }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Iterate thru all the tasks

A for-each loop also works, though the forEach() method is more "modern"

Applying a One-shot Phaser with Java

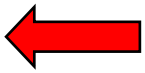
- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                  entryPhaser.register();
                  new Thread(() -> {
                               entryPhaser.arriveAndAwaitAdvance();

                               task.run();
                               exitPhaser.arrive();
                             }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```



Dynamically add a new party to the entry barrier

This capability is not available with a CyclicBarrier

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();

                new Thread(() -> {
                  entryPhaser.arriveAndAwaitAdvance();

                  task.run();
                  exitPhaser.arrive();
                }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Create/start a new worker thread that runs the task once other threads arrive

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                  entryPhaser.register();

                  new Thread(() -> {
                                  entryPhaser.arriveAndAwaitAdvance();

                                  task.run();
                                  exitPhaser.arrive();
                                }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Block until all worker threads have started

Phaser.arriveAndAwaitAdvance() is similar to CyclicBarrier.await()

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                  entryPhaser.register();

                  new Thread(() -> {
                                  entryPhaser.arriveAndAwaitAdvance();

                                  task.run();
                                  exitPhaser.
                                 }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Block until all worker threads have started

This code uses entryPhaser as a one-shot "entry barrier"

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();

                new Thread(() -> {
                          entryPhaser.arriveAndAwaitAdvance();

                          task.run();
                          exitPhaser.arrive();
                         }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Run the task

The details of what MyTask does are not important for our discussion

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();

                new Thread(() -> {
                           entryPhaser.arriveAndAwaitAdvance();

                           task.run();
                           exitPhaser.arrive();
                }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Indicate the task/ thread is done

Phaser.arrive() is used similarly to CountDownLatch.countdown() here

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

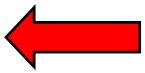
```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();

                new Thread(() -> {
                           entryPhaser.arriveAndAwaitAdvance();

                           task.run();
                           exitPhaser.arrive();
                          }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```



Allow worker threads to start running their tasks

Applying a One-shot Phaser with Java

- Shows one-shot Phasers that start running a list of tasks simultaneously

```
void runOneShotTasks(List<MyTask> tasks) {
  Phaser entryPhaser = new Phaser(1);
  Phaser exitPhaser = new Phaser(tasks.size());

  tasks.forEach(task -> {
                entryPhaser.register();

                new Thread(() -> {
                           entryPhaser.arriveAndAwaitAdvance();

                           task.run();
                           exitPhaser.arrive();
                          }).start();
    });

  entryPhaser.arriveAndDeregister();
  exitPhaser.awaitAdvance(0);
}
```

Block until all worker threads have finished

Phaser.awaitAdvance() is used similarly to CountDownLatch.await() here

Applying a Cyclic Phaser with Java

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Create a phaser that runs a given # of times

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Determines when to terminate phaser

onAdvance() is a hook method

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Terminates when all iterations have completed or all parties are done

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};
  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Bulk register calling thread & all the tasks!

Phaser.bulkRegister() is used in lieu of registering each task in forEach()

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                  do {
                    task.run();
                    phaser.arriveAndAwaitAdvance();
                  } while (!phaser.isTerminated());
                  }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Iterate thru all the tasks

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                  do {
                    task.run();
                    phaser.arriveAndAwaitAdvance();
                  } while (!phaser.isTerminated());
                  }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Create/start a worker thread

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                  do {
                    task.run();
                    phaser.arriveAndAwaitAdvance();
                  } while (!phaser.isTerminated());
                  }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Run the task

The details of what MyTask does are not important for our discussion

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                  do {
                    task.run();
                    phaser.arriveAndAwaitAdvance();
                  } while (!phaser.isTerminated());
                  }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Block until all the other threads/tasks complete this phase

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() ->
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

This code is using the phaser as a "cyclic exit barrier"

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

The last thread to arrive at the end of a phase triggers a call to the onAdvance() hook method

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};
  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Terminate when the phase # + 1 == iterations

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Threads loop until they are terminated by onAdvance()

Applying Cyclic Phaser with Java

- A cyclic Phaser that repeatedly performs actions for a given # of iterations

```
void runCyclicTasks(List<MyTask> tasks, int iterations) {
  Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase,
                                int regParties) {
      return (phase + 1) == iterations || regParties == 0;
  }};

  phaser.bulkRegister(1 + tasks.size());

  tasks.forEach(task -> { new Thread(() -> {
                do {
                  task.run();
                  phaser.arriveAndAwaitAdvance();
                } while (!phaser.isTerminated());
                }).start();
  });
  while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

Calling thread loops until terminated by onAdvance()

arriveAndAwaitAdvance() blocks waiting the end of each phase, so this loop does not "busy wait"

End of Example Application of Java Phaser