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

```java
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](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

```java
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

```java
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](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26)
Main entry point into the test program

```java
private static List<MyTask> makeTasks() {
    return IntStream.rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());
}
```

```java
static void main(String[] argv) {
    runOneShotTasks(makeTasks());
    runCyclicTasks(makeTasks(), sITERATIONS);
}
```

However, the details of what MyTask does are not important for our discussion.
• Main entry point into the test program

```java
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 stream from 1 to sNUMBER_OF_TASKS
• Main entry point into the test program

```java
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);
    }
Test Driver Program Walkthrough

- Main entry point into the test program

```java
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"
• Main entry point into the test program

```java
private static List<MyTask> makeTasks() {
    return IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());
}
```

```java
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

```java
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/douglascairnschmidt/LiveLessons/tree/master/Java8/ex26
Applying a One-shot Phaser with Java

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

```java
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 “entry barrier” Phaser is similar to a CyclicBarrier (but more flexible)
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 “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

```java
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);
}
```

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

Iterate thru all the tasks
Applying a One-shot Phaser with Java

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

```java
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 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

```java
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

```java
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);
}
```

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

```java
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 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

```java
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 `task.run();`  
`exitPhaser.arrive();`  
`()).start();`

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

```java
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);
}
```

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

```java
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

```java
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);
}
```

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

```java
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](https://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

```java
void runCyclicTasks(List<MyTask> tasks, int iterations) {
    Phaser phaser = new Phaser() { // Create a phaser that runs a given # of times
        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](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

```java
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();
}
```

onAdvance() is a hook method

Determines when to terminate phaser
Applying Cyclic Phaser with Java

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

```java
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](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex26)
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();
}
Applying Cyclic Phaser with Java

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

```java
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 number of iterations.

```java
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();
}
```

Create/start a worker thread.
Applying Cyclic Phaser with Java

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

```java
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 {
            Run the task task.run();
            phaser.arriveAndAwaitAdvance();
        } while (!phaser.isTerminated());
    }).start();

    while (!phaser.isTerminated()) phaser.arriveAndAwaitAdvance();
}
```

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

```java
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 number of iterations

```java
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"
A cyclic Phaser that repeatedly performs actions for a given # of iterations

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

```java
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**
A cyclic Phaser that repeatedly performs actions for a given number of iterations.

```java
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

```java
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();
}
```

Calling thread loops until terminated by onAdvance() blocks waiting the end of each phase, so this loop does not “busy wait”
End of Example Application of Java Phaser