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

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```

This program is based on examples in the Java documentation available 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)
Test Driver Program Walkthrough
Test Driver Program Walkthrough

• Main entry point into the test program

```java
static void main(String[] argv) {

    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());

    runTasks(tasks);

    startTasks(tasks, sITERATIONS);
}
```

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex26](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex26)
• Main entry point into the test program

```java
static void main(String[] argv) {
    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());

    runTasks(tasks);

    startTasks(tasks, sITERATIONS);
}
```

Create a list of MyTask objects

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

```java
static void main(String[] argv) {
    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());

    runTasks(tasks);

    startTasks(tasks, sITERATIONS);
}
```

Create a stream from 1 to sNUMBER_OF_TASKS
Main entry point into the test program

```java
static void main(String[] argv) {
    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());

    runTasks(tasks);

    startTasks(tasks, sITERATIONS);
}
```

Create a new MyTask object for each number in the stream
Test Driver Program Walkthrough

- Main entry point into the test program

```java
static void main(String[] argv) {

    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());  // Convert the stream into a list of MyTask objects

    runTasks(tasks);

    startTasks(tasks, sITERATIONS);
}
```
Main entry point into the test program

```java
static void main(String[] argv) {

    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());

    runTasks(tasks);  // Run the test showcasing a one-shot Phaser that runs a list of tasks that all start at the same time

    startTasks(tasks, sITERATIONS);
}
```

This method illustrates the use of a Phaser as an “entry barrier”
Test Driver Program Walkthrough

• Main entry point into the test program

```java
static void main(String[] argv) {
    List<MyTask> tasks = IntStream
        .rangeClosed(1, sNUMBER_OF_TASKS)
        .mapToObj(MyTask::new)
        .collect(toList());

    runTasks(tasks);  // Run the test that showcases a cyclic Phaser that repeatedly performs actions for a given # of iterations

    startTasks(tasks, sITERATIONS);
}
```

This method illustrates the use of a Phaser as a “cyclic barrier”
Applying a One-shot Phaser with Java
• A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```
Applying a One-shot Phaser with Java

- A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```

Create a new phaser with a “parties” value of 1 to implicitly register itself.
Applying a One-shot Phaser with Java

- A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```
Applying a One-shot Phaser with Java

• A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

    tasks.forEach(task -> {
        phaser.register();
        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```

Inform phaser there’s a new party to add
Applying a One-shot Phaser with Java

- A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```
Applying a One-shot Phaser with Java

A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```
Applying a One-shot Phaser with Java

- A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```

Block until all worker threads have started

This code is using the phaser as a one-shot “entry barrier”
Applying a One-shot Phaser with Java

• A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```
Applying a One-shot Phaser with Java

- A one-shot Phaser that starts running a list of tasks simultaneously

```java
void runTasks(List<MyTask> tasks) {
    Phaser phaser = new Phaser(1);

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

        new Thread(() -> {
            phaser.arriveAndAwaitAdvance();
            task.run();
        }).start();
    });

    phaser.arriveAndDeregister();
}
```

Allow thread calling `runTasks()` to continue & deregister itself so worker threads can start running their tasks
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 startTasks(List<MyTask> tasks, int iterations) {
    Phaser phaser = new Phaser() {
        protected boolean onAdvance(int phase, int regParties) {
            return phase >= iterations || regParties == 0;
        }
    };

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

Create a new phaser that iterates a given # of times
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

This hook method determines when to terminate the phaser.
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

Terminate the phaser when all iterations have completed
Applying Cyclic Phaser with Java

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

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

    phaser.register();  // Register to defer worker threads advancing to second phase until the end of this method

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

    phaser.arriveAndDeregister();
}
```
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

Block until all other tasks/threads complete this phase
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

Block until all other tasks/threads complete this phase

This code is using the phaser as a “cyclic barrier”
A cyclic Phaser that repeatedly performs actions for a given # of iterations

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

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

32
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

The phaser is terminated when the phase # is $\geq$ the iterations param
Applying Cyclic Phaser with Java

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

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

    phaser.register();

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

    phaser.arriveAndDeregister();
}
```

Loop until phaser is terminated by onAdvance()
Applying Cyclic Phaser with Java

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

```java
void startTasks(List<MyTask> tasks, int iterations) {
    Phaser phaser = new Phaser() {
        protected boolean onAdvance(int phase, int regParties) {
            return phase >= iterations || regParties == 0;
        }
    };
    phaser.register();
    tasks.forEach(task -> {
        phaser.register();
        new Thread(() -> {
            do {
                task.run();
                phaser.arriveAndAwaitAdvance();
            } while (!phaser.isTerminated());
        }).start();
    });
    phaser.arriveAndDeregister();
}
```

Deregister itself (allowing tasks to advance to next phase) & don't wait
End of Java Phaser: Example Application
1. Why is it ok for notifyAll() to be called prior to removing/returning the first item in the queue?
   a. Because the monitor lock is still held & thus only one method can run in the monitor object
   b. Because notifyAll() induces the “Thundering Herd” problem, which delays notified threads from waking up before the first item in the queue is removed/returned
   c. Because the associated monitor lock is released automatically when wait() is called
   d. Because there’s just a single wait queue in a Java built-in monitor object