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
Key Methods in Java Phaser
Key Methods in Java Phaser

- Phaser has a more complex API than CountDownLatch or CyclicBarrier
  - i.e., it has many methods that support a range of use cases
Key Methods in Java Phaser

- Phaser has a more complex API than CountDownLatch or CyclicBarrier
  - i.e., it has many methods that support a range of use cases

Fortunately, many of these methods are rarely used in practice.
Key Methods in Java Phaser

- Constructor creates a new object with an initial phase # of 0

```java
public class Phaser {
    ...
    public Phaser(int parties) {
        ...
    }

    public Phaser() { ... }
    ...
}
```
Key Methods in Java Phaser

- Constructor creates a new object with an initial phase # of 0
- This constructor specifies the # of parties needed to advance to the next phase

```java
public class Phaser {
    ... public Phaser(int parties) {
        ...}
    public Phaser() { ... }
    ...}
```

# of registered parties dictates when a phaser can advance to the next phase
Key Methods in Java Phaser

- Constructor creates a new object with an initial phase # of 0
  - This constructor specifies the # of parties needed to advance to the next phase
  - This constructor is optional since parties can always register later

```java
public class Phaser {
    ...  
    public Phaser(int parties) {
        ...  
    }
    public Phaser() { ... }  
    ...  
}
```

With Java Phaser the # of parties need not match the # of threads
Key Methods in Java Phaser

- Constructor creates a new object with an initial phase # of 0
  - This constructor specifies the # of parties needed to advance to the next phase
  - This constructor doesn’t specify any parties initially

```java
public class Phaser {
    ... 
    public Phaser(int parties) {
        ... 
    }
    public Phaser() { ... } 
    ... 
}
```
Key Methods in Java Phaser

- Constructor creates a new object with an initial phase # of 0
  - This constructor specifies the # of parties needed to advance to the next phase
  - This constructor doesn’t specify any parties initially
    - Any thread using a phaser created via this constructor therefore needs to register with it before using it

```java
public class Phaser {
    ...
    public Phaser(int parties) {
        ...
    }
    public Phaser() { ... }
    ...
```
Key Methods in Java Phaser

- Constructor creates a new object with an initial phase # of 0
- This constructor specifies the # of parties needed to advance to the next phase
- This constructor doesn’t specify any parties initially
- Any thread using a phaser created via this constructor therefore needs to register with it before using it

```java
public class Phaser {
    ...
    public Phaser(int parties) {
        ...
    }
    ...
    public Phaser() { ... } ...
}
```
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate

```java
public class Phaser {
    ...
    public int register() { ... }

    public int bulkRegister(int parties) { ... }

    public int arriveAndAwaitAdvance() {
        ... }

    public int ArriveAndDeregister() {
        ... }

    protected boolean onAdvance(int phase, int registeredParties) {
        return registeredParties == 0;
    }
```
Phaser’s key methods enable parties to register, synchronize, & terminate.

- Adds unarrived parties to phaser

```java
public class Phaser {
    ...
    public int register() { ... }
    public int bulkRegister(int parties) { ... }
}
```

# of registered parties dictates when a phaser can advance to the next phase.
Phaser’s key methods enable parties to register, synchronize, & terminate

- Adds unarrived parties to phaser
- Arrive & await advance

Having multiple methods provides flexibility wrt arrival & waiting to advance
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
    - Arrives at phaser, but does not block until other parties arrive

```java
public class Phaser {
    ...
    public int arrive() { ... }
    
    public int awaitAdvance(int phase)
    { ... }
    
    public int arriveAndAwaitAdvance()
    { ... }
}
```
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
    - Arrives at phaser, but does not block until other parties arrive
    - Returns arrival phase # or a negative value if the phaser has already terminated

```java
public class Phaser {
    ...
    public int arrive() { ... }
    public int awaitAdvance(int phase) {
        ... }
    public int arriveAndAwaitAdvance() {
        ... }
}
```

This method is rarely used in practice
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
    - Arrives at phaser, but does not block until other parties arrive
    - Blocks until the phase of this phaser advances from the given phase value

```java
public class Phaser {
    ...
    public int arrive() { ... }

    public int awaitAdvance(int phase) {
        ... }

    public int arriveAndAwaitAdvance() {
        ... }
}
```
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
    - Arrives at phaser, but does not block until other parties arrive
    - Blocks until the phase of this phaser advances from the given phase value
    - Returns immediately if current phrase != given phase

```java
public class Phaser {  
    ...  
    public int arrive() { ... }  
    public int awaitAdvance(int phase) { ... }  
    public int arriveAndAwaitAdvance() { ... }  
}
```

This method is rarely used in practice
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
    - Arrives at phaser, but does not block until other parties arrive
    - Blocks until the phase of this phaser advances from the given phase value
    - Arrives at phaser & blocks until other parties arrive

```java
public class Phaser {
    ...
    public int arrive() { ... }
    public int awaitAdvance(int phase) {
        ...
    }
    public int arriveAndAwaitAdvance() {
        ...
    }

    Equivalent in effect to awaitAdvance(arrive())
```
Phaser’s key methods enable parties to register, synchronize, & terminate

- Adds unarrived parties to phaser
- Arrive & await advance
  - Arrives at phaser, but does not block until other parties arrive
  - Blocks until the phase of this phaser advances from the given phase value
  - Arrives at phaser & blocks until other parties arrive

This method is commonly used & is similar to `await()` on a Java CyclicBarrier
Phaser’s key methods enable parties to register, synchronize, & terminate

- Adds unarrived parties to phaser
- Arrive & await advance
- Arrive at the phaser & deregister without waiting for others to arrive

```java
public class Phaser {
    ...
    public int arriveAndDeregister() {
        ...
    }
```
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
- Adds unarrived parties to phaser
- Arrive & await advance
- Arrive at the phaser & deregister without waiting for others to arrive
- Reduces # of parties required to advance in future phases

```java
public class Phaser {
    ...
    public int arriveAndDeregister() {
        ...
    }
}
```

Often used by the party that controls the initialization of a Phaser
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
  - Arrive at the phaser & deregister without waiting for others to arrive
  - Hook method performs an action upon pending phase advance

```java
public class Phaser {
    ...
    protected boolean onAdvance (int phase, int registeredParties) {
        return registeredParties == 0;
    }
}
```

*This method is invoked upon arrival of the party advancing the phaser*
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
- Adds unarrived parties to phaser
- Arrive & await advance
- Arrive at the phaser & deregister without waiting for others to arrive
- Hook method performs an action upon pending phase advance

```java
public class Phaser {
    ...
    protected boolean onAdvance
        (int phase,
         int registeredParties) {
        return registeredParties == 0;
    }
```

This hook method is similar to the barrier action on a Java CyclicBarrier
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
- Adds unarrived parties to phaser
- Arrive & await advance
- Arrive at the phaser & deregister without waiting for others to arrive
- Hook method performs an action upon pending phase advance
- Also initiates termination by returning a ‘true’ Boolean value

```java
public class Phaser {
    ...
    protected boolean onAdvance(
        int phase,
        int registeredParties)
    {
        return registeredParties == 0;
    }
}
```
Key Methods in Java Phaser

- Phaser’s key methods enable parties to register, synchronize, & terminate
  - Adds unarrived parties to phaser
  - Arrive & await advance
  - Arrive at the phaser & deregister without waiting for others to arrive
  - Hook method performs an action upon pending phase advance
  - Also initiates termination by returning a ‘true’ Boolean value

```java
public class Phaser {
    ...
    protected boolean onAdvance(
        int phase,
        int registeredParties)
    {
        return registeredParties == 0;
    }
}
```

Default implementation terminates if there are no registered parties
End of Java Phaser: Key Methods
1. What of the following are benefit of the Java Phaser over the CyclicBarrier?
   
a. *It supports fixed-size “cyclic” & “entry” and/or “exit” barriers who # of parties match the # of threads*

b. *It supports variable-size “cyclic” & “entry” and/or “exit” barriers whose # of parties can vary dynamically*

c. *It uses the AbstractQueuedSynchronizer framework to enhance reuse*

d. *They provide better support for fixed-sized # of parties*