Key Methods in Java
ForkJoinTask Superclass

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

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Recognize the key methods in the ForkJoinPool class

• Recognize the key methods in the ForkJoinTask class
Key Methods in Java ForkJoinTask
Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future

```java
abstract class ForkJoinTask<V>
    implements Future<V>,
    Serializable {
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html)
Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future

abstract class ForkJoinTask<V> implements Future<V>, Serializable {
...

It’s uncommon to use these future methods, but rather use subclass methods.
There are three key methods:
- `fork()` enables a task to create sub-tasks that run in parallel.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#fork](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#fork)
There are three key methods

- fork() enables a task to create sub-tasks that run in parallel
- Arrange to execute this task asynchronously in the current task’s pool or ForkJoinPool’s common pool

```
abstract class ForkJoinTask<V>
    implements Future<V>, Serializable {
    ...
    final ForkJoinTask<V> fork() {
        ...
    }

    final V join() {
        ...
    }

    final V invoke() {
        ...
    }
```
There are three key methods

- `fork()` enables a task to create sub-tasks that run in parallel
- Arrange to execute this task asynchronously in the current task’s pool or ForkJoinPool’s common pool

The `fork()` method does not block the caller

```java
abstract class ForkJoinTask<V> implements Future<V>, Serializable {
    ...
    final ForkJoinTask<V> fork() {
        ... }
    final V join() { ... }
    final V invoke() { ... }
}
```
There are three key methods:

- **fork()** enables a task to create sub-tasks that run in parallel.
- Arrange to execute this task asynchronously in the current task’s pool or ForkJoinPool’s common pool.
- Pushes the task on the head of the deque owned by the current worker thread.

<table>
<thead>
<tr>
<th>WorkQueue</th>
<th>WorkQueue</th>
<th>WorkQueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-Task(_{1,1})</td>
<td>Sub-Task(_{1,2})</td>
<td>Sub-Task(_{3,3})</td>
</tr>
<tr>
<td>Sub-Task(_{1,3})</td>
<td>Sub-Task(_{1,4})</td>
<td>Sub-Task(_{3,4})</td>
</tr>
</tbody>
</table>

1. **fork()**
2. **push()**
Key Methods in Java ForkJoinTask

- There are three key methods
  - `fork()` enables a task to create sub-tasks that run in parallel
  - `join()` returns the result of a previously fork’d computation when it’s done

abstract class `ForkJoinTask<V>`
  implements `Future<V>`,
  `Serializable`

  ... final `ForkJoinTask<V>` `fork()`
  { ... }

  final `V` `join()` { ... }

  final `V` `invoke()` { ... }

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#join
There are three key methods:

- **fork()** enables a task to create sub-tasks that run in parallel.
- **join()** returns the result of a previously fork’d computation when it’s done.
- Calling task is “blocked” until forked sub-task is done.

```java
abstract class ForkJoinTask<V>
    implements Future<V>, Serializable {

    ...

    final ForkJoinTask<V> fork() { ... }

    final V join() { ... }

    final V invoke() { ... }
```
There are three key methods
- fork() enables a task to create sub-tasks that run in parallel
- join() returns the result of a previously fork’d computation when it’s done
- Calling task is “blocked” until forked sub-task is done

abstract class ForkJoinTask<V>
    implements Future<V>, Serializable {
    ...
    final ForkJoinTask<V> fork() {
        ...
    }

    final V join() {
        ...
    }

    final V invoke() {
        ...
    }

"Collaborative Jiffy Lube” model of processing!

See [en.wikipedia.org/wiki/Jiffy_Lube](en.wikipedia.org/wiki/Jiffy_Lube)
Key Methods in Java ForkJoinTask

- There are three key methods
  - `fork()` enables a task to create sub-tasks that run in parallel
  - `join()` returns the result of a previously fork’d computation when it’s done
    - Calling task is “blocked” until forked sub-task is done
  - Defines a synchronization point

```java
abstract class ForkJoinTask<V>
    implements Future<V>,
    Serializable {
    ...
    final ForkJoinTask<V> fork()
        { ... } 

    final V join() { ... } 

    final V invoke() { ... } 
```

Key Methods in Java ForkJoinTask

- There are three key methods
  - fork() enables a task to create sub-tasks that run in parallel
  - join() returns the result of a previously fork'd computation when it’s done
  - Calling task is “blocked” until forked sub-task is done
- Defines a synchronization point
- Ensures all writes in a worker thread that “happened-before” join() are made visible to other threads after the join()

abstract class ForkJoinTask<V>
  implements Future<V>, Serializable {
    ...
    final ForkJoinTask<V> fork() {
      ...
    }
    final V join() {
      ...
    }
    final V invoke() {
      ...
    }

See www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/happens-before.html
There are three key methods:

- `fork()` enables a task to create sub-tasks that run in parallel.
- `join()` returns the result of a previously fork’d computation when it’s done.
- `invoke()` performs this task, awaits its completion if needed, & returns its result.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#invoke](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#invoke)
Key Methods in Java ForkJoinTask

- There are three key methods
  - `fork()` enables a task to create sub-tasks that run in parallel
  - `join()` returns the result of a previously fork’d computation when it’s done
  - `invoke()` performs this task, awaits its completion if needed, & returns its result
- Throws `RuntimeException` or `Error` if the underlying computation did so

```java
abstract class ForkJoinTask<V> implements Future<V>, Serializable {
    ...
    final ForkJoinTask<V> fork() {
        ...
    }
    final V join() { ...
    }
    final V invoke() { ...
    }
}
```

See www.baeldung.com/java-exceptions
End of Key Methods in the Java ForkJoinTask Superclass