Applying Java Functional Programming Features to Start & Join Threads

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

• Understand how Java functional programming features are applied in a simple parallel program

• Know how to start & join Java threads via functional programming features
Example of Starting & Joining Java Threads
Example of Starting & Joining Java Threads

- Showcases Java FP features

```java
public void run() {
    // Code here
}
```

See [ThreadJoinTest/updated/src/main/java/ThreadJoinTest.java](ThreadJoinTest/updated/src/main/java/ThreadJoinTest.java)
Example of Starting & Joining Java Threads

• Showcases Java FP features

```java
public void run() {
    // Start a group of threads that search for phrases in parallel
    "MacBeth" "Hamlet" "King Lear" "Julius Caesar"
```

Start a group of threads that search for phrases in parallel
Example of Starting & Joining Java Threads

- Showcases Java FP features

```java
public void run() {
  // Code goes here
}
```

A Java thread is a unit of computation that runs in the context of a process.

See [docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html](docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html)
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
- Flexibly create worker threads via a factory method

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads(this::processInput);
    ...
}
```

Factory method makes a list of worker threads
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads(
        this::processInput);
    ...
}

Void processInput(String input) {
    ...}

List<Thread> makeWorkerThreads(
    Function<String, Void> task) {
    ...}
```

This method searches for phrases in one work of Shakespeare
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads(
        this::processInput);
    ...
}

Void processInput(String input) {
    ...}

List<Thread> makeWorkerThreads(
    Function<String, Void> task)
    { ... }
```

This functional interface makes it easy to change the function passed to factory method
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Apply a function lambda to create runnable for a thread

```java
List<Thread> makeWorkerThreads(Function<String, Void> task) {
    List<Thread> workerThreads = new ArrayList<>();
    mInputList.forEach(input ->
        workerThreads.add(new Thread(() -> task.apply(input))));

    return workerThreads;
}
```

This factory method creates a list of threads that will be joined when their processing is done.
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Apply a function lambda to create runnable for a thread

```java
List<Thread> makeWorkerThreads(
    Function<String, Void> task)
{
    List<Thread> workerThreads =
        new ArrayList<>();

    mInputList.forEach(input ->
        workerThreads.add(new Thread(()
            -> task.apply(input)));

    return workerThreads;
}
```

Create an empty list of threads
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
- Flexibly create worker threads via a factory method
- Pass a reference to a method expecting a functional interface
- Apply a function lambda to create runnable for a thread

```java
List<Thread> makeWorkerThreads(
    Function<String, Void> task)
{
    List<Thread> workerThreads =
        new ArrayList<>();

    mInputList.forEach(input ->
        workerThreads.add(new Thread(()
            -> task.apply(input)));

    return workerThreads;
}
```

Create a thread for each input string to perform processing designated by the task parameter
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Apply a function lambda to create runnable for a thread

```java
List<Thread> makeWorkerThreads(
    Function<String, Void> task) {
    List<Thread> workerThreads =
        new ArrayList<>();

    mInputList.forEach(input ->
        workerThreads.add(
            new Thread(
                () -> task.apply(input))));

    return workerThreads;
}
```

`task.apply()` creates a runnable that provides the computation for each of the threads
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Apply a function lambda to create runnable for a thread

```java
List<Thread> makeWorkerThreads(Function<String, Void> task) {
    List<Thread> workerThreads = new ArrayList<>();

    mInputList.forEach(input ->
        workerThreads.add(new Thread(() -> task.apply(input))));

    return workerThreads;
}
```

*Add each new thread to the list*
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Apply a function lambda to create runnable for a thread

```java
List<Thread> makeWorkerThreads(Function<String, Void> task) {
    List<Thread> workerThreads = new ArrayList<>();

    mInputList.forEach(input ->
        workerThreads.add(new Thread(() -> task.apply(input))));

    return workerThreads;
}
```

Return the list of worker threads
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Start worker threads via `forEach()` & a method reference

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads
                                (this::processInput);

    workerThreads
        .forEach(Thread::start);
    ...
}
```

Each worker thread has its own runtime call stack

See [en.wikipedia.org/wiki/Call_stack](https://en.wikipedia.org/wiki/Call_stack)
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Start worker threads via `forEach` & a method reference

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads
        (this::processInput);

    workerThreads.forEach(Thread::start);

    // ... forEach() & method reference
    start each worker thread to search for phrases in works of Shakespeare
```

Hamlet  Macbeth
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Start worker threads via `forEach()` & a method reference

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads
        (this::processInput);

    workerThreads
        .forEach(Thread::start);
    ...
}
```

This program uses a “thread-per-work” parallelism model
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Start worker threads via forEach() & a method reference
  - Wait for worker threads to finish

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads
        (this::processInput);

    workerThreads
        .forEach(Thread::start);

    workerThreads
        .forEach(thread -> {
            ... thread.join(); ...
        } ...)

    Uses forEach() & lambda expression
```
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Start worker threads via forEach() & a method reference
- Wait for worker threads to finish

```
public void run() {
    List<Thread> workerThreads =
        makeWorkerThreads
        (this::processInput);

    workerThreads
        .forEach(Thread::start);

    workerThreads
        .forEach(thread -> {
            ... thread.join(); ...
        } ...)
```

See en.wikipedia.org/wiki/Barrier_(computer_science)
Example of Starting & Joining Java Threads

- Showcases Java FP features, e.g.
  - Flexibly create worker threads via a factory method
  - Pass a reference to a method expecting a functional interface
  - Start worker threads via `forEach()` & a method reference
  - Wait for worker threads to finish

```java
public void run() {
    List<Thread> workerThreads = makeWorkerThreads
        (this::processInput);

    workerThreads
        .forEach(Thread::start);

    workerThreads
        .forEach(thread -> {
            ... thread.join(); ...
        } ...)

    No other Java synchronizers are needed!
```
End of Applying Java Functional Programming Features to Start & Join Threads