Motivating the Need for Java 8 CompletableFutures

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

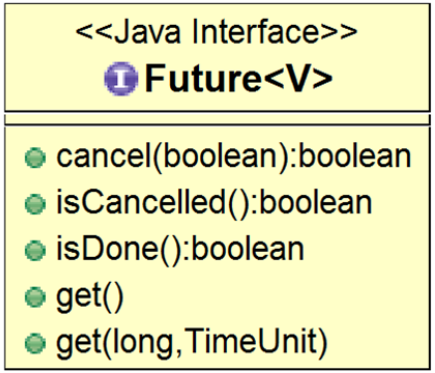
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

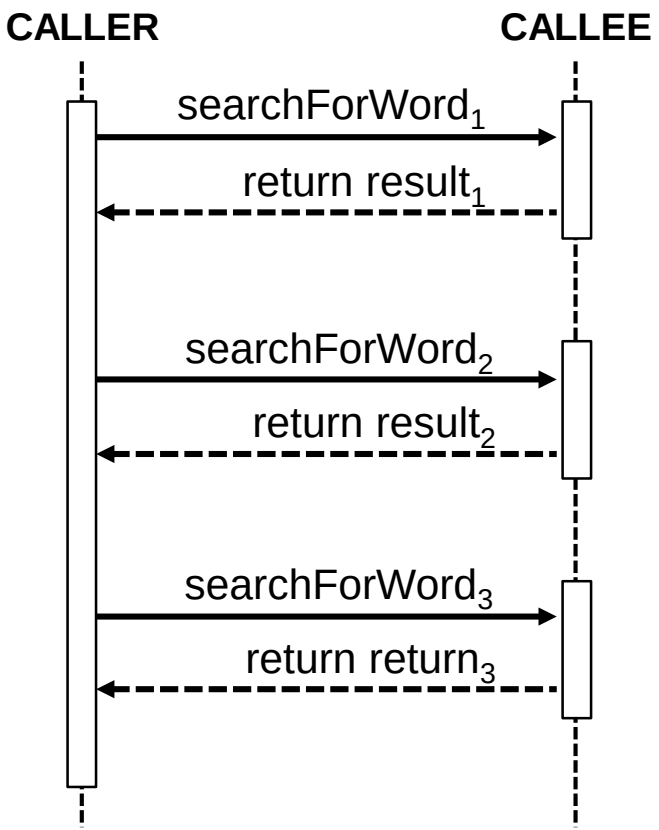
- Motivate the need for Java 8 completable futures



Motivating the Need for Completable Futures

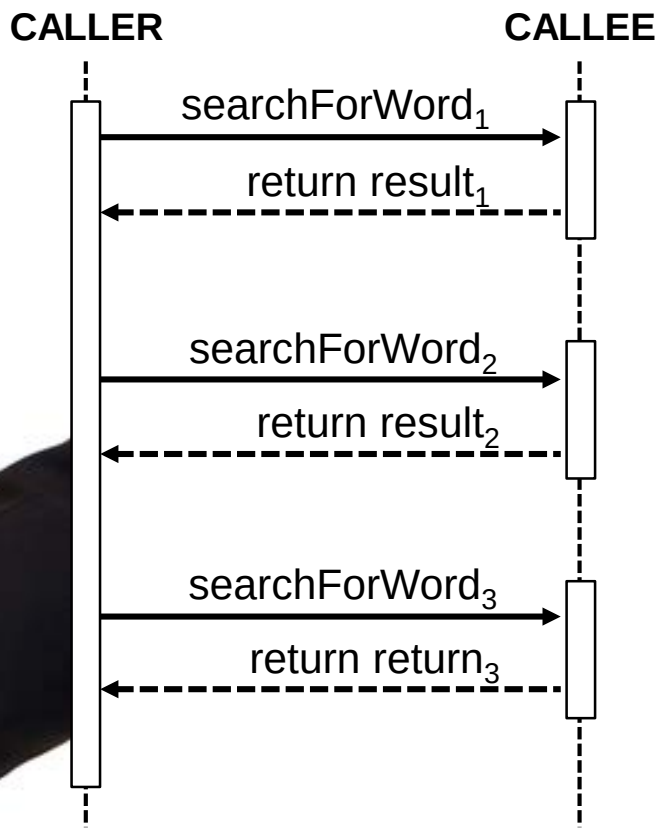
Motivating the Need for Completable Futures

- Thus far, behaviors running in aggregate operations have all been synchronous



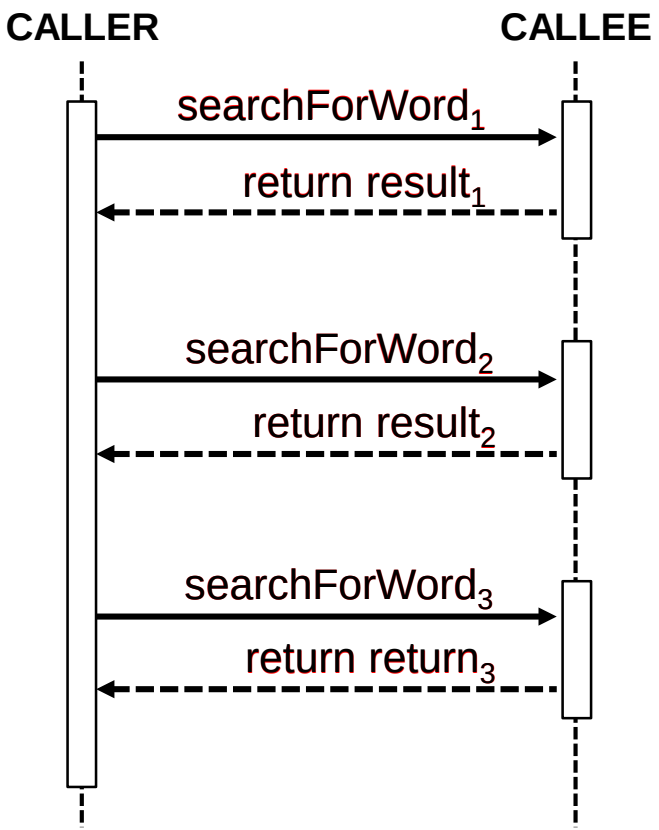
Motivating the Need for Completable Futures

- Thus far, behaviors running in aggregate operations have all been synchronous
  - A synchronous behavior borrows the thread of its caller until its computation(s) finish



Motivating the Need for Completable Futures

- Thus far, behaviors running in aggregate operations have all been synchronous
  - A synchronous behavior borrows the thread of its caller until its computation(s) finish

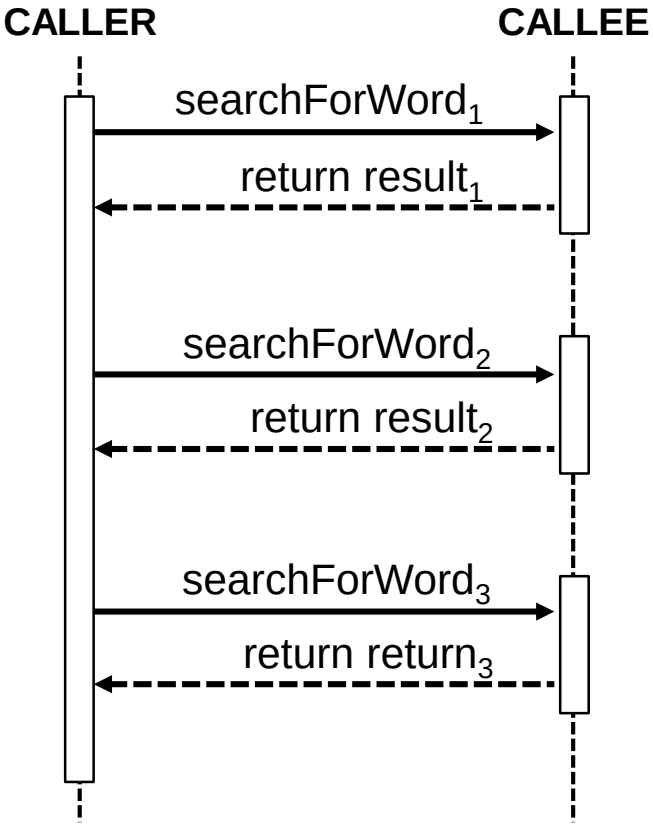


Motivating the Need for Completable Futures

- Pros & cons of synchronous calls

Motivating the Need for Completable Futures

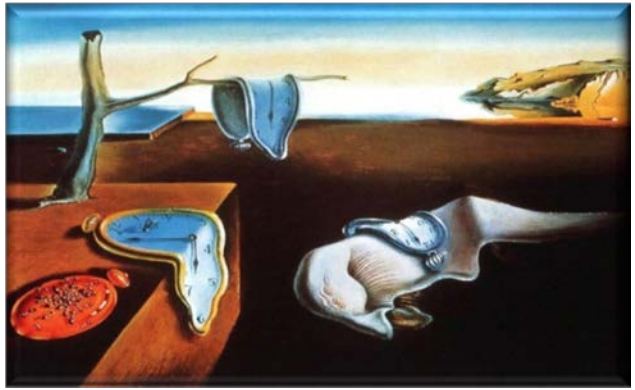
- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns

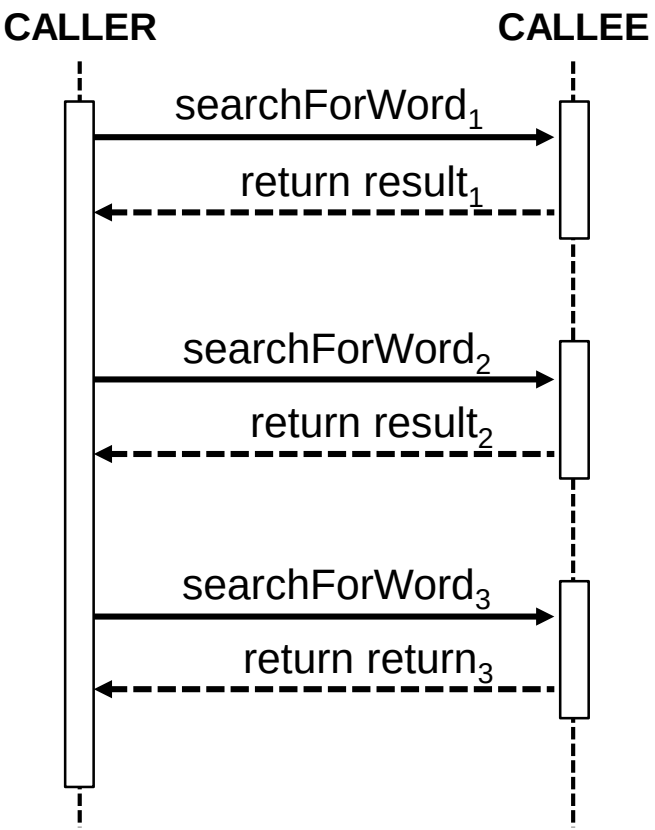


See www.iro.umontreal.ca/~keller/Layla/remote.pdf

Motivating the Need for Completable Futures

- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns
  - *Cons*:
    - May not leverage all the parallelism available in multi-core systems



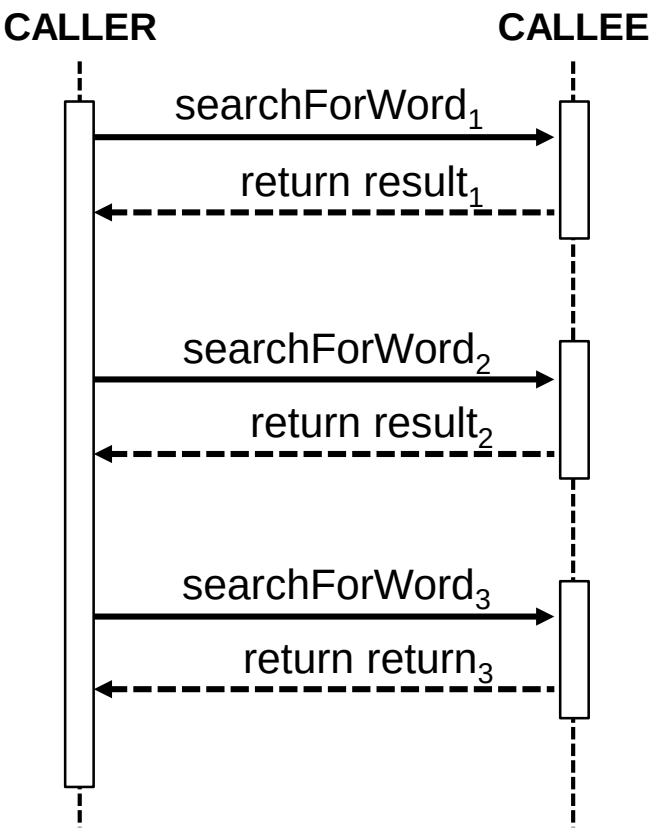


Motivating the Need for Completable Futures

- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns
  - *Cons*:
    - May not leverage all the parallelism available in multi-core systems, e.g.
      - Selecting number of threads is hard

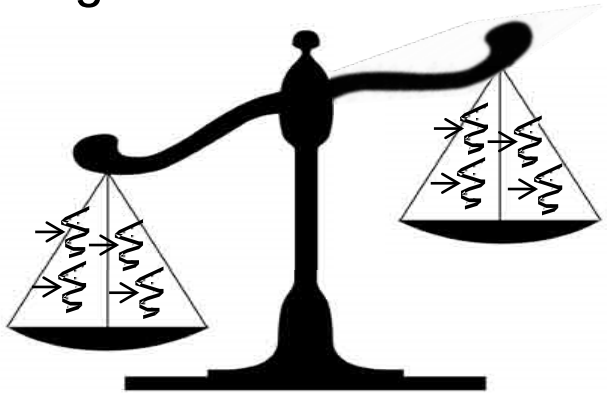
Efficient Performance

Efficient Resource Utilization



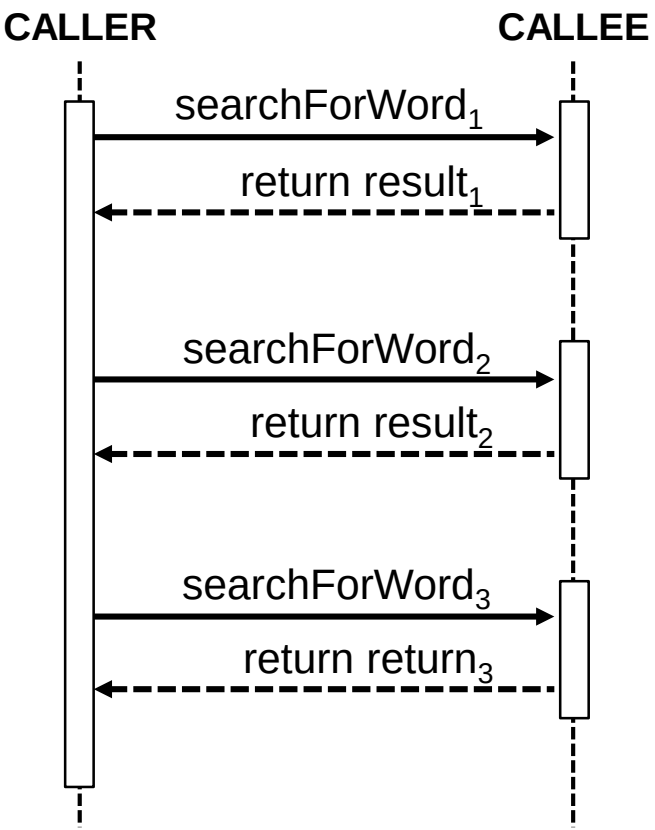
Motivating the Need for Completable Futures

- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns
  - *Cons*:
    - May not leverage all the parallelism available in multi-core systems, e.g.
      - Selecting number of threads is hard



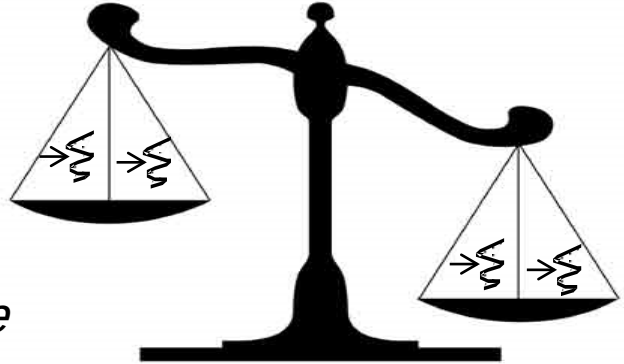
Efficient Performance

Efficient Resource Utilization



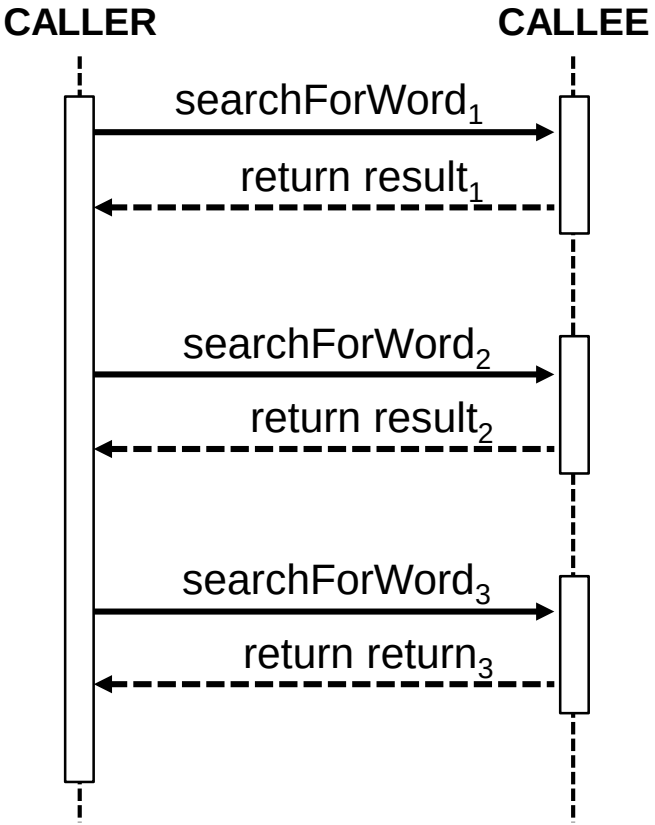
Motivating the Need for Completable Futures

- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns
  - *Cons*:
    - May not leverage all the parallelism available in multi-core systems, e.g.
      - Selecting number of threads is hard



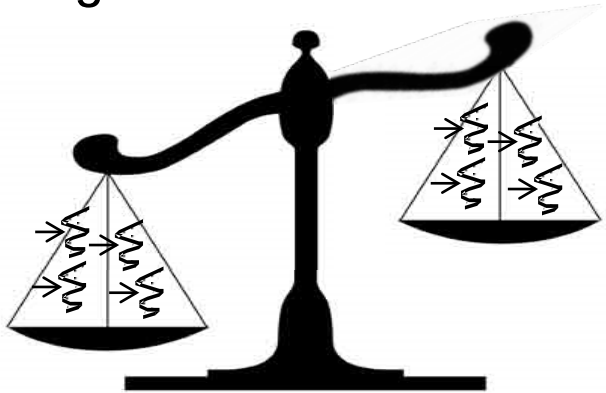
Efficient Performance

Efficient Resource Utilization



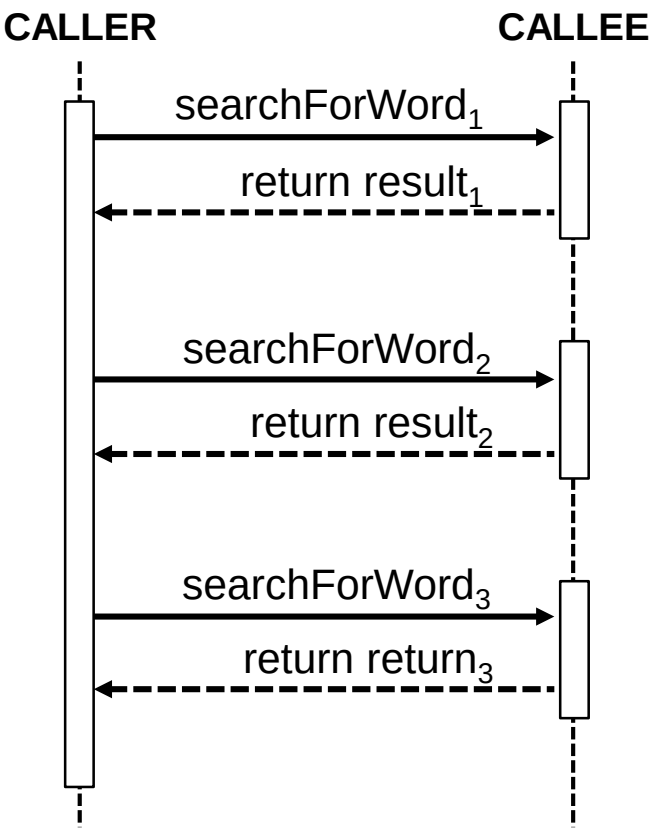
Motivating the Need for Completable Futures

- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns
  - *Cons*:
    - May not leverage all the parallelism available in multi-core systems, e.g.
      - Selecting number of threads is hard



Efficient Performance

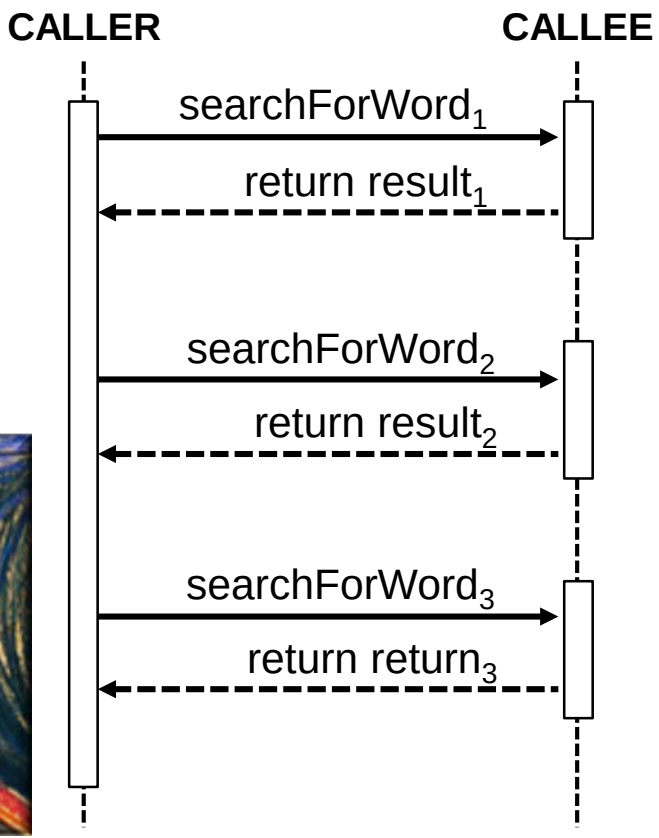
Efficient Resource Utilization



Particularly tricky for I/O-bound programs that need more threads to run efficiently

Motivating the Need for Completable Futures

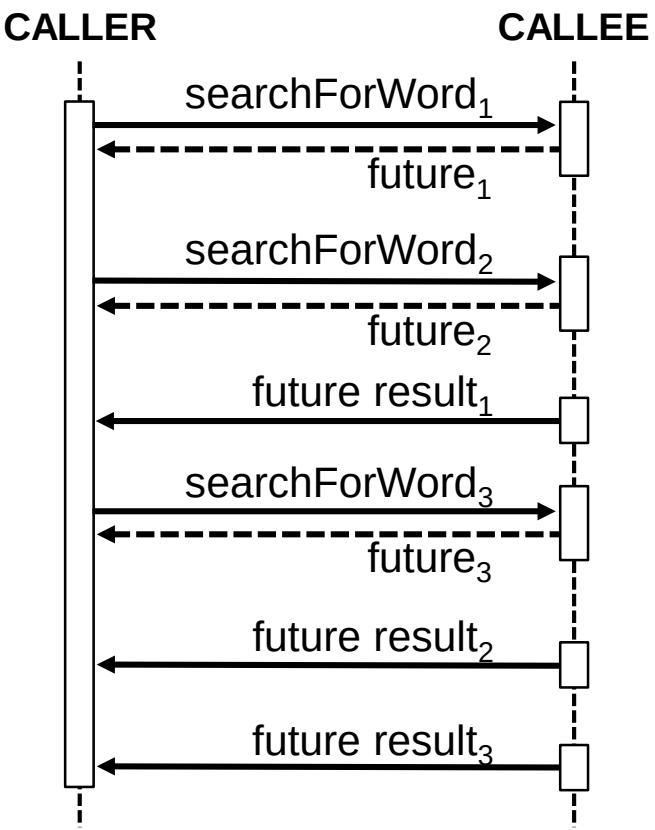
- Pros & cons of synchronous calls
  - *Pros*: "Intuitive" since they map cleanly onto conventional two-way method patterns
  - *Cons*:
    - May not leverage all the parallelism available in multi-core systems
    - Synchronous calls may need to (dynamically) change the size of the common fork-join pool



See dzone.com/articles/think-twice-using-java-8

Motivating the Need for Completable Futures

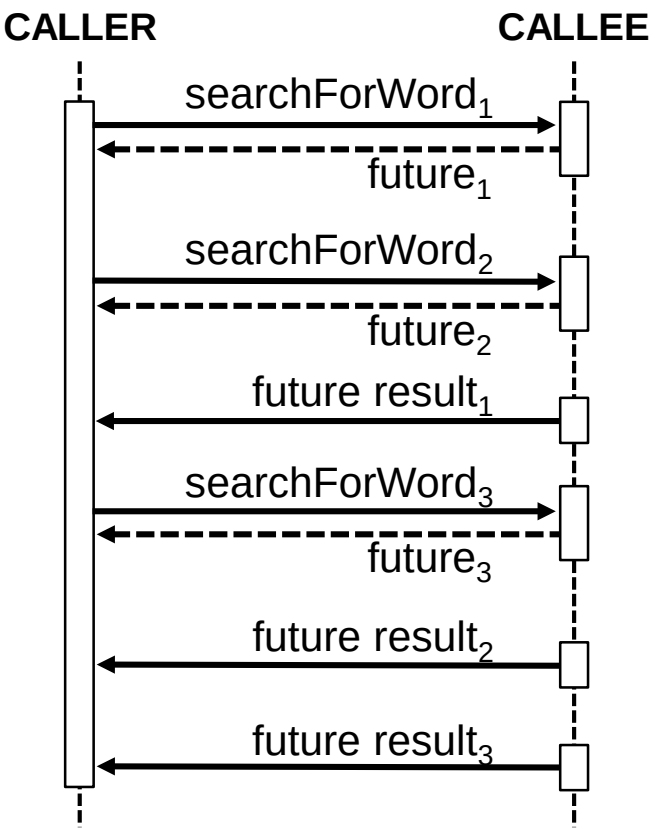
- Another approach uses asynchronous calls & Java futures



See en.wikipedia.org/wiki/Asynchrony_(computer_programming)

Motivating the Need for Completable Futures

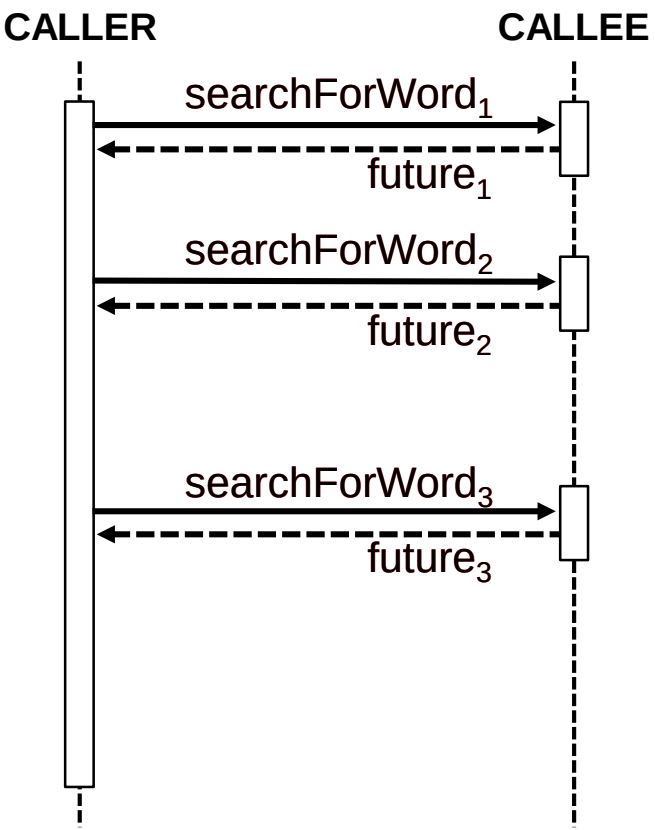
- Another approach uses asynchronous calls & Java futures



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Motivating the Need for Completable Futures

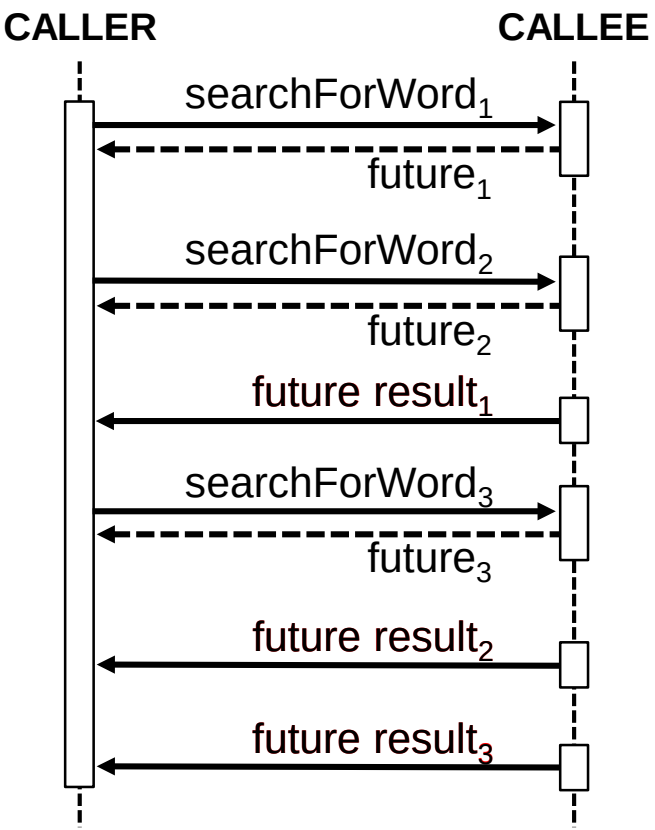
- Another approach uses asynchronous calls & Java futures
  - Asynchronous calls return a future immediately & continue running the computation in the background



See en.wikipedia.org/wiki/Asynchrony_(computer_programming)

Motivating the Need for Completable Futures

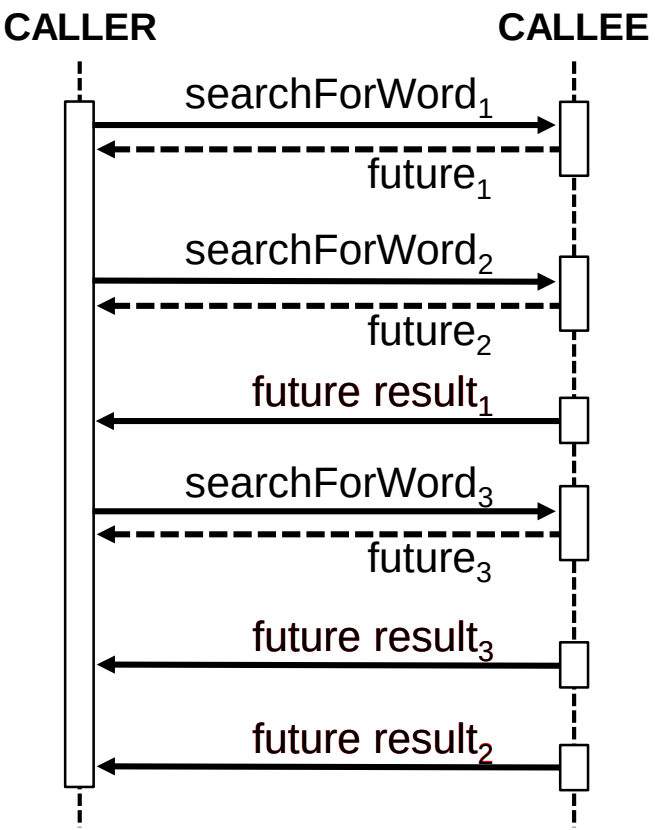
- Another approach uses asynchronous calls & Java futures
  - Asynchronous calls return a future immediately & continue running the computation in the background
  - When the computation completes the future is triggered & the caller can get the result



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Motivating the Need for Completable Futures

- Another approach uses asynchronous calls & Java futures
  - Asynchronous calls return a future immediately & continue running the computation in the background
  - When the computation completes the future is triggered & the caller can get the result
    - Results can occur in a different order than the original calls were made



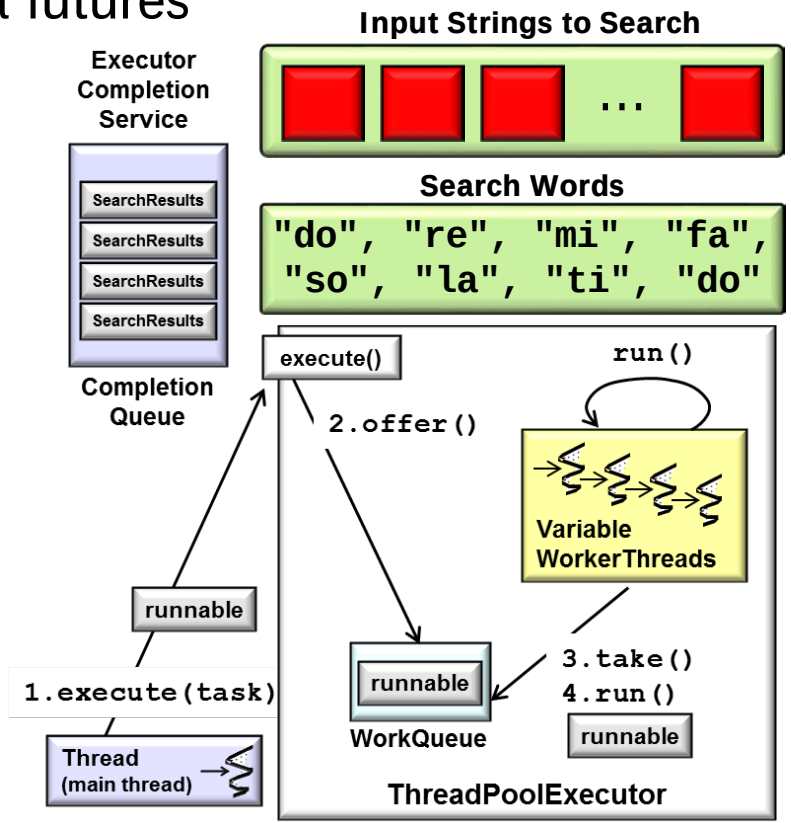
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures

Motivating the Need for Completable Futures

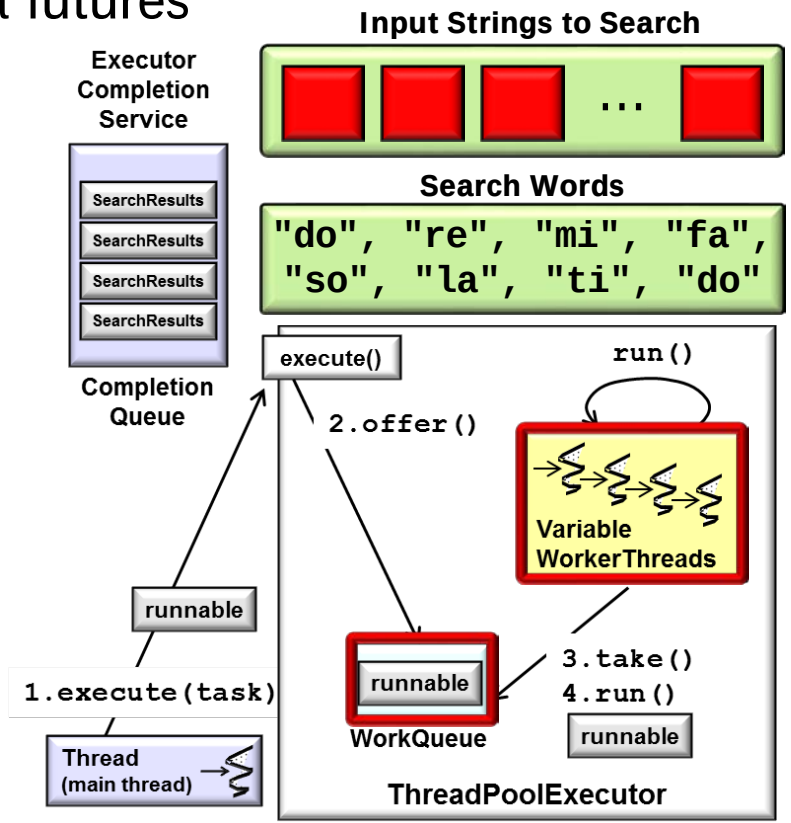
- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Can leverage inherent parallelism more effectively



See Lesson 2.3 on the Java Executor Framework

Motivating the Need for Completable Futures

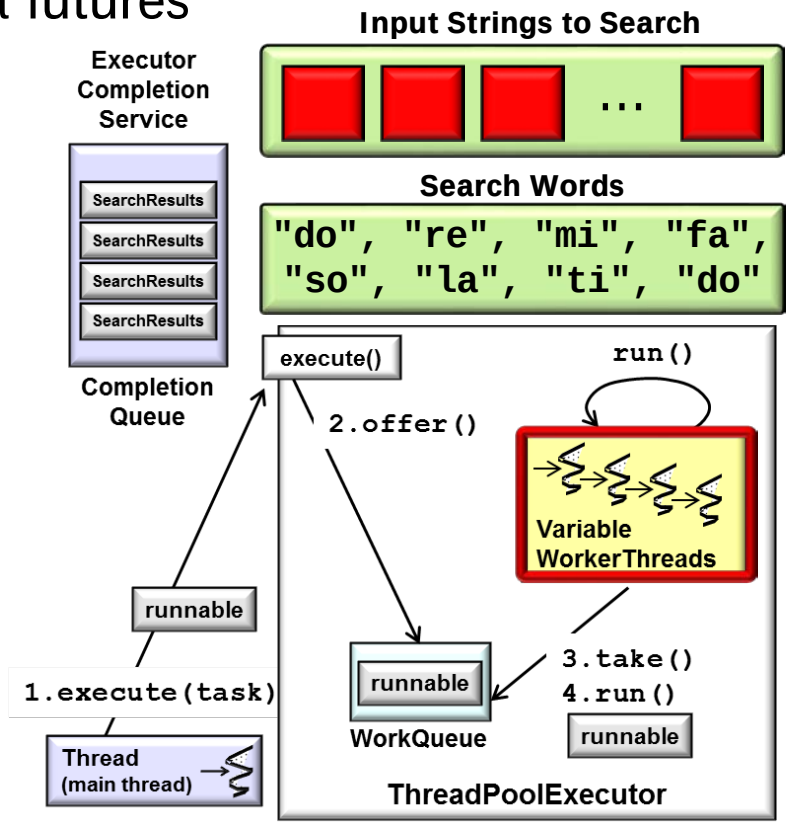
- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Can leverage inherent parallelism more effectively, e.g.,
    - Queue asynchronous computations for execution in a pool of threads



See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchTaskGang

Motivating the Need for Completable Futures

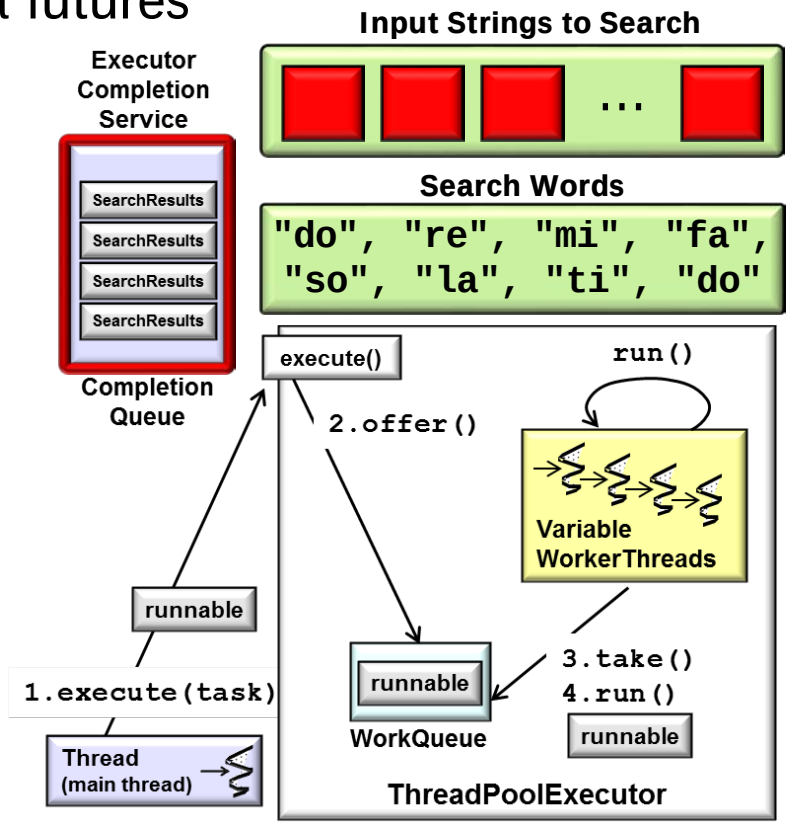
- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Can leverage inherent parallelism more effectively, e.g.,
    - Queue asynchronous computations for execution in a pool of threads
    - Automatically tune variable number of threads based on the workload



See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchTaskGang

Motivating the Need for Completable Futures

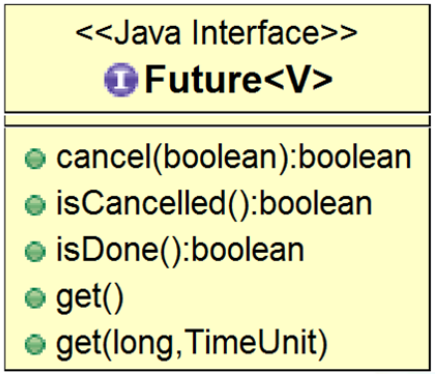
- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Can leverage inherent parallelism more effectively, e.g.,
    - Queue asynchronous computations for execution in a pool of threads
    - Automatically tune variable number of threads based on the workload
    - Array of futures can be triggered to get the results



See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchTaskGang

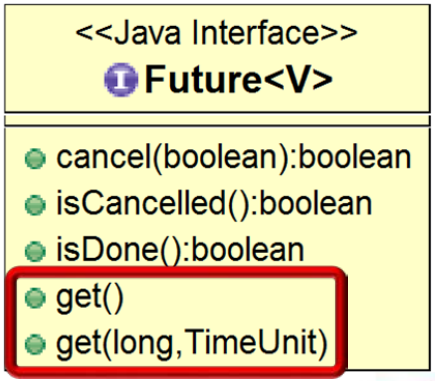
Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Leverage inherent parallelism more effectively with fewer threads
  - *Cons*: Limited in their features



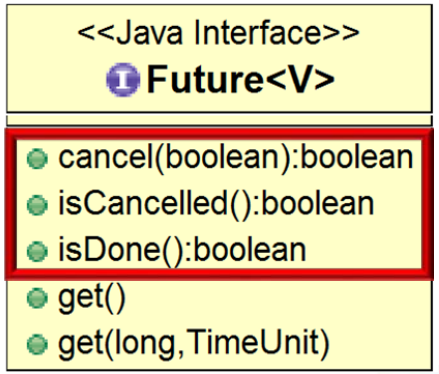
Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Leverage inherent parallelism more effectively with fewer threads
  - *Cons*: Limited in their features, e.g.,
    - *Can* return the result of an async two-way task



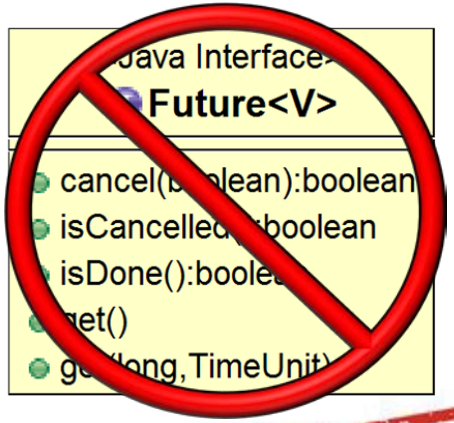
Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Leverage inherent parallelism more effectively with fewer threads
  - *Cons*: Limited in their features, e.g.,
    - *Can* return the result of an async two-way task
    - *Can* be canceled & tested to see if a task is done



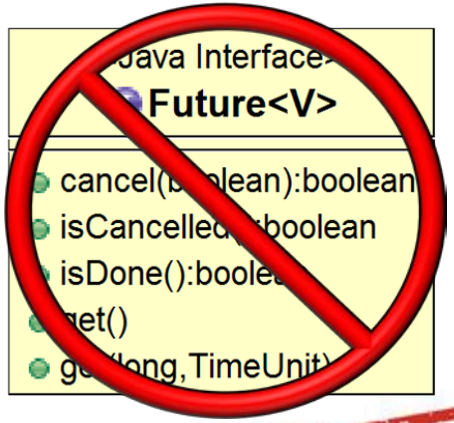
Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Leverage inherent parallelism more effectively with fewer threads
  - *Cons*: Limited in their features, e.g.,
    - *Can* return the result of an async two-way task
    - *Can* be canceled & tested to see if a task is done
    - *Cannot* be completed explicitly



Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Leverage inherent parallelism more effectively with fewer threads
  - *Cons*: Limited in their features, e.g.,
    - Can return the result of an async two-way task
    - Can be canceled & tested to see if a task is done
    - *Cannot* be completed explicitly
    - *Cannot* be triggered reactively/ efficiently as a *set* of futures w/out extra effort & overhead



It's awkward & inefficient to try & "compose" multiple futures

Motivating the Need for Completable Futures

- Pros & cons of asynchronous calls with Java futures
  - *Pros*: Leverage inherent parallelism more effectively with fewer threads
  - *Cons*: Limited in their features, e.g.,
    - Can return the result of an async two-way task
    - Can be canceled & tested to see if a task is done
    - *Cannot* be completed explicitly
    - *Cannot* be triggered reactively/ efficiently as a *set* of futures w/out extra effort & overhead

Java 8 completable futures are designed to overcome these limitations with futures

End of Motivating the Need for Java 8 Completable Futures