Overview of Java 8 CompletableFutures (Part 3)

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 the basic completable futures features
- Understand another advanced completable futures feature

Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

`CompletionStage<T>, Future<T>`

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

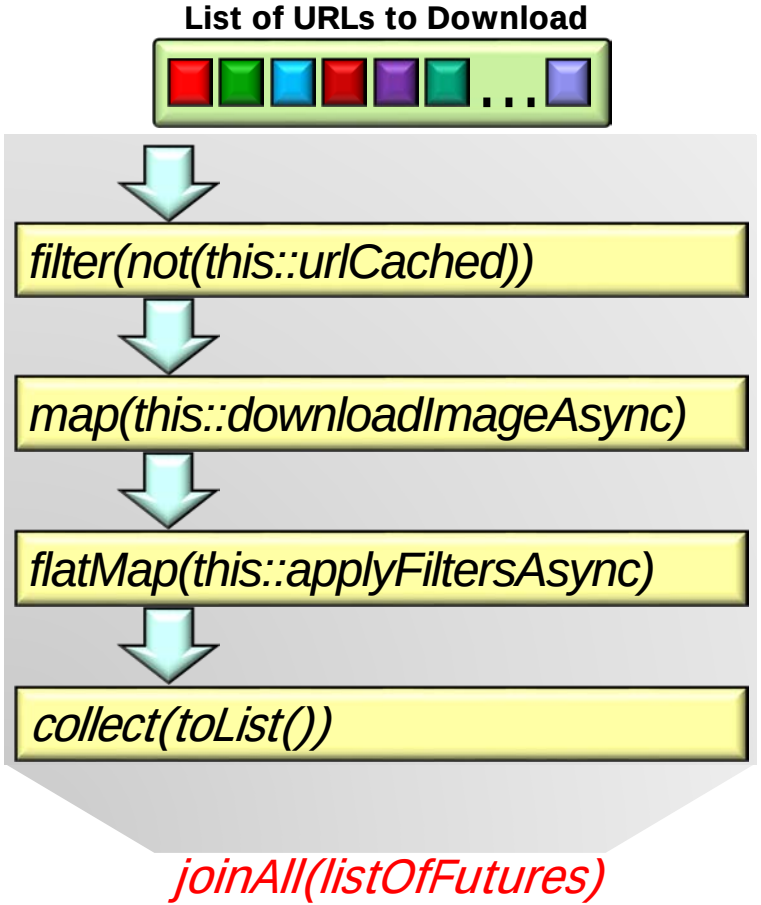
A `Future` that may be explicitly completed (setting its value and status), and may be used as a `CompletionStage`, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to `complete`, `completeExceptionally`, or `cancel` a CompletableFuture, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, CompletableFuture implements interface `CompletionStage` with the following policies:

Learning Objectives in this Part of the Lesson

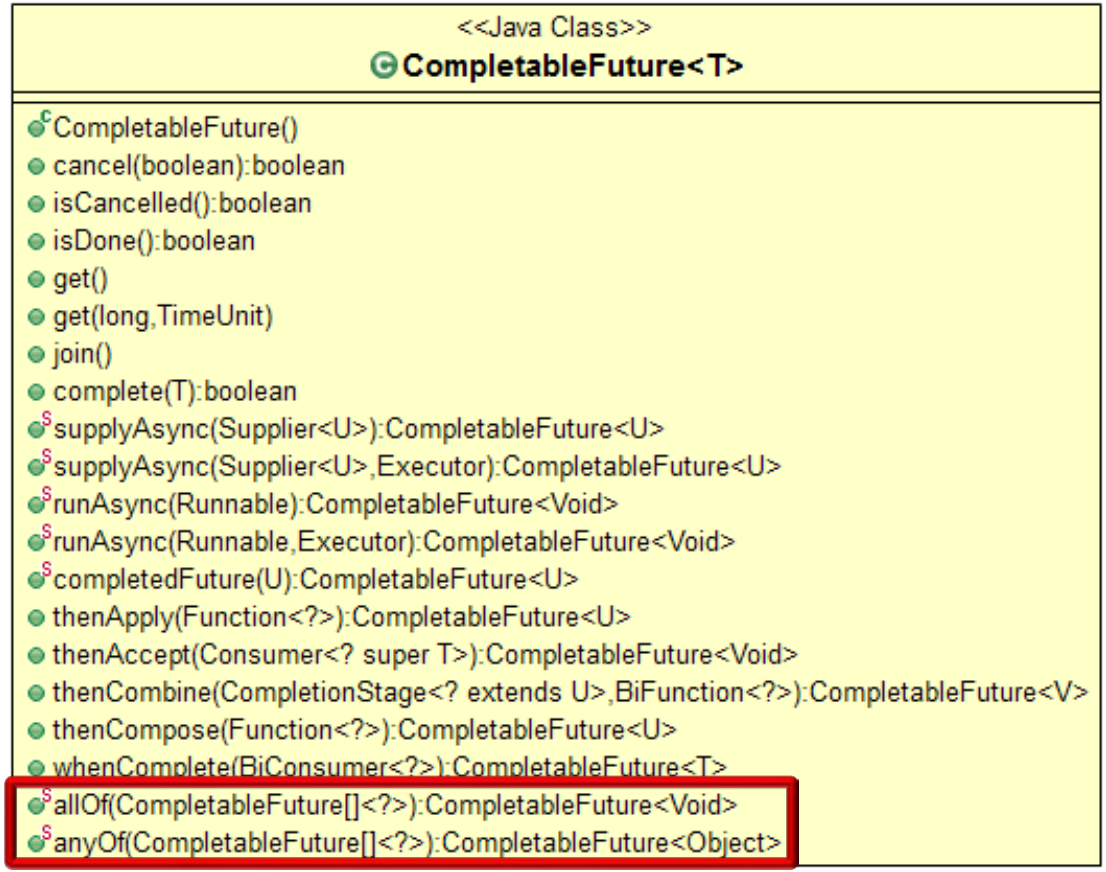
- Understand the basic completable futures features
- Understand another advanced completable futures features
  - A method from a completable futures implementation of ImageStreamGang is used as an example



Arbitrary-Arity Methods

Arbitrary-Arity Methods

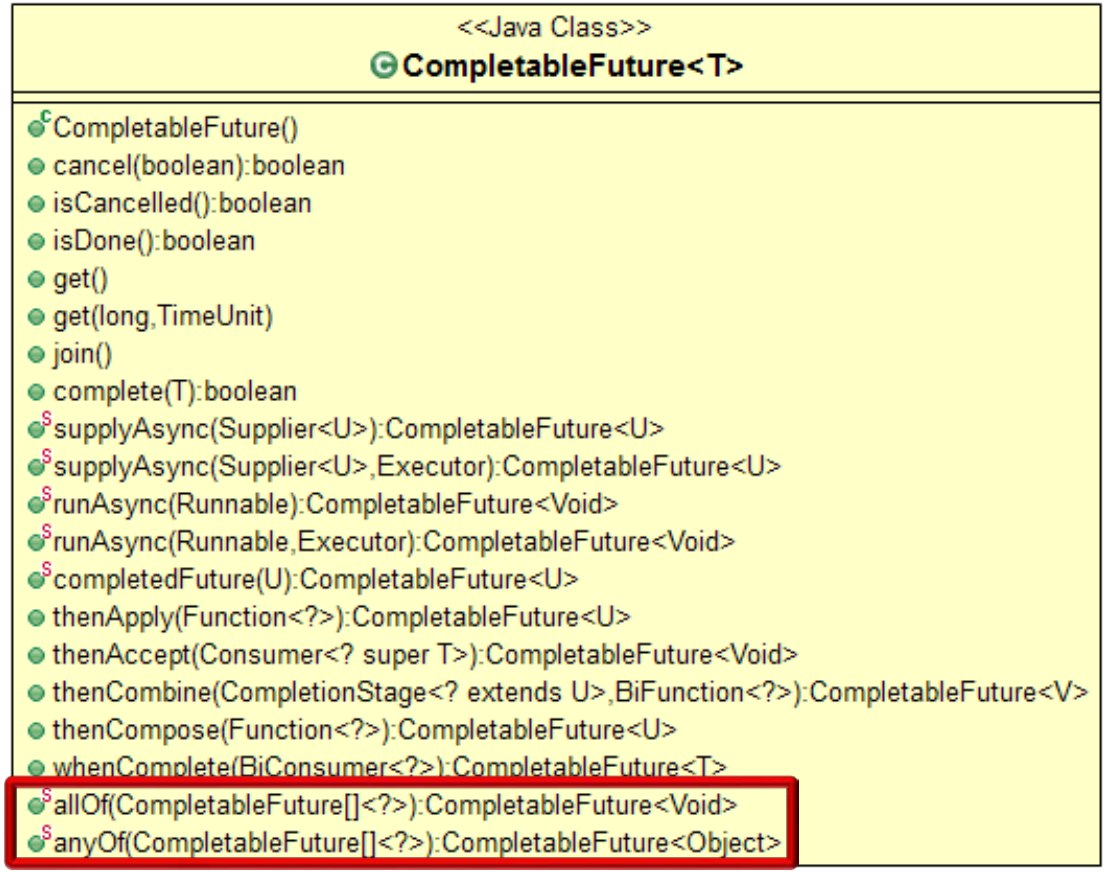
- Completable future also support “arbitrary-arity” methods



See en.wikipedia.org/wiki/Arity

Overview of Completable Futures: Advanced Features

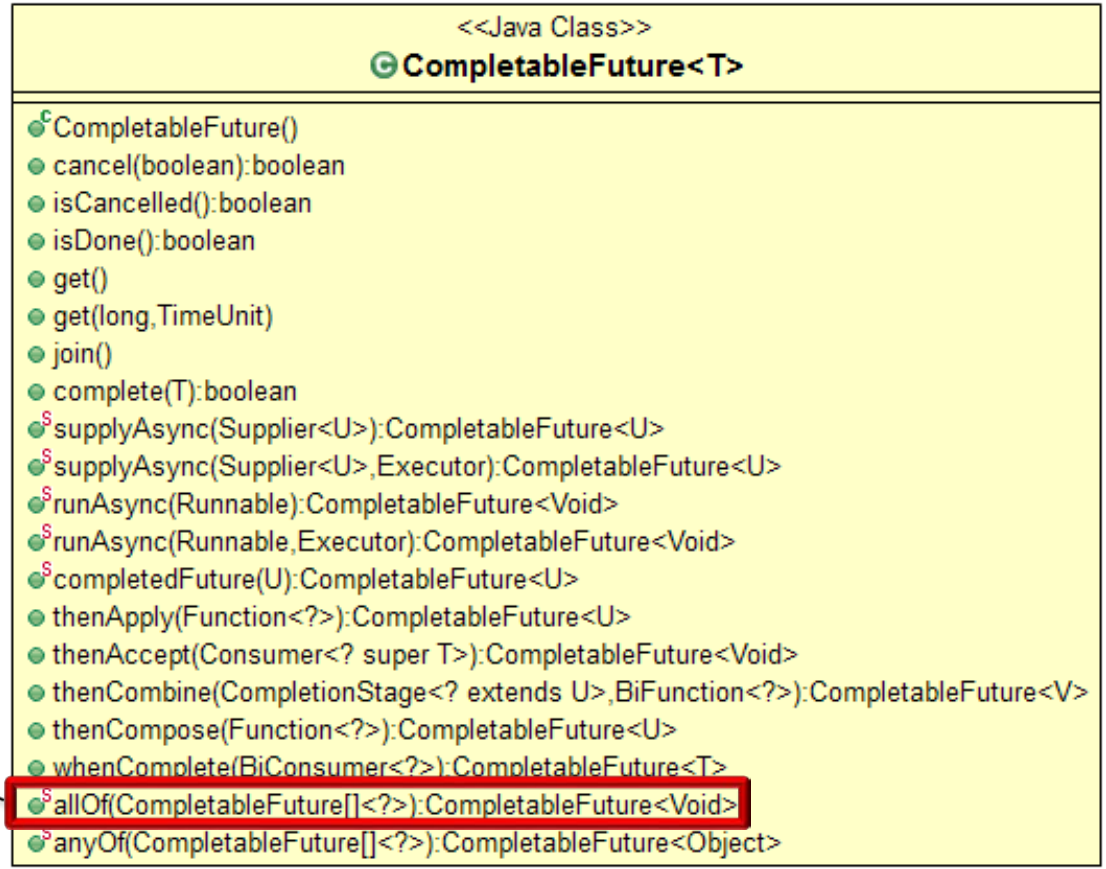
- Completable future also support “arbitrary-arity” methods
  - Can wait for any or all completable futures in an array to complete



Overview of Completable Futures: Advanced Features

- Completable future also support “arbitrary-arity” methods
  - Can wait for any or all completable futures in an array to complete

We focus on allOf()



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));
  return dList;
}
```

See ImageStreamGang/AndroidGUI/app/src/main/java/livelessons/utils/StreamUtils.java

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

The return value converts a list of completed futures into a list of joined results

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));
  return dList;
}
```

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

The parameter is a list of completable futures to some generic type T

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));
  return dList;
}
```

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
         .stream()
         .map(CompletableFuture::join)
         .collect(toList()));
  return dList;
}
```

Returns a completable future that completes when all completable futures complete

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));
  return dList;
}
```

Create an array that stores the list of completable futures

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
         .stream()
         .map(CompletableFuture::join)
         .collect(toList()));
  return dList;
}
```

Creates a completable future to a list of joined results when all completable futures complete

Overview of Completable Futures: Advanced Features

- The StreamUtils.joinAll() method provides a useful wrapper that encapsulates allOf()

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
         .stream()
         .map(CompletableFuture::join)
         .collect(toList()));
  return dList;
}
```

Return a completable future to the list of joined results

Overview of Completable Futures: Advanced Features

- joinAll() provides a very powerful wrapper for some complex code!!!

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
          fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
         .stream()
         .map(CompletableFuture::join)
         .collect(toList()));
  return dList;
}
```

See www.nurkiewicz.com/2013/05/java-8-completablefuture-in-action.html

Overview of Completable Futures: Advanced Features

- joinAll() provides a very powerful wrapper for some complex code!!!

```
static <T> CompletableFuture<List<T>>
joinAll(List<CompletableFuture<T>>
        fList) {
  CompletableFuture<Void>
    dFuture = CompletableFuture.allOf
      (fList.toArray(new
       CompletableFuture[fList.size()]));

  CompletableFuture<List<T>> dList =
    dFuture.thenApply(v -> fList
        .stream()
        .map(CompletableFuture::join)
        .collect(toList()));
  return dList;
}
```

Also see www.nurkiewicz.com/2013/05/java-8-definitive-guide-to.html

End of Overview of Java 8 Completable Futures (Part 3)