

Concurrent and Parallel

Preliminary

The following questions are not going to be discussed. Answers is given without explanation. You are encouraged to discuss them on Ed.

1. Consider the following code snippet. In this implementation, `busy()` may run for a different arbitrary amount of time in each call.

   Code Snippet

   ```
   CompletableFuture<Void> a = CompletableFuture.runAsync(() -> {
     busy();
     System.out.print("A");
   });
   CompletableFuture<Void> b = a.thenRun(() -> {
     busy();
     System.out.print("B");
   });
   CompletableFuture<Void> c = b.thenRun(() -> {
     busy();
     System.out.print("C");
   });
   CompletableFuture<Void> d = c.thenRun(() -> {
     busy();
     System.out.print("D");
   });
   ```

   (a) Draw the dependency diagram between `a`, `b`, `c`, and `d`.

   (b) Find all possible print output from the code snippet above such that all characters `A`, `B`, `C`, and `D` are printed.

2. Consider the following code snippet. In this implementation, `busy()` may run for a different arbitrary amount of time in each call.

   Code Snippet

   ```
   CompletableFuture<Void> a = CompletableFuture.runAsync(() -> {
     busy();
     System.out.print("A");
   });

   CompletableFuture<Void> b = a.thenRun(() -> {
   ```

```
  busy();
  System.out.print("B");
});
CompletableFuture<Void> c = a.thenRun(() -> {
  busy();
  System.out.print("C");
});
CompletableFuture<Void> d = a.thenRun(() -> {
  busy();
  System.out.print("D");
});
```

(a) Draw the dependency diagram between `a`, `b`, `c`, and `d`.

(b) Find all possible print output from the code snippet above such that all characters `A`, `B`, `C`, and `D` are printed.

3. Consider the following code snippet. In this implementation, `busy()` may run for a different arbitrary amount of time in each call.

Code Snippet

```
CompletableFuture<Void> a = CompletableFuture.runAsync(() -> {
  busy();
  System.out.print("A");
});
CompletableFuture<Void> b = a.thenRun(() -> {
  busy();
  System.out.print("B");
});
CompletableFuture<Void> c = a.thenRun(() -> {
  busy();
  System.out.print("C");
});
CompletableFuture<Void> d = c.runAfterBoth(b, () -> {
  busy();
  System.out.print("D");
});
```

(a) Draw the dependency diagram between `a`, `b`, `c`, and `d`.

(b) Find all possible print output from the code snippet above such that all characters `A`, `B`, `C`, and `D` are printed.

Discussion

The following questions are problem sets to be discussed during recitation. You are expected to attempt them before coming to the recitation. You may be randomly called to present your answer during recitation. You are encouraged to discuss them on Ed.

1. Consider the program below using `CompletableFuture` [1].

A.java

```
import java.util.concurrent.CompletableFuture;

class A {
  private final int x;
```

```
  public A(int x) {
    this.x = x;
  }

  private void sleep() {
    System.out.println(Thread.currentThread().getName() + " " + x);
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      System.out.println("interrupted");
    }
  }

  public A incr() {
    sleep();
    return new A(this.x + 1);
  }

  public A decr() {
    sleep();
    if (x <= 0) {
      throw new IllegalStateException();
    }
    return new A(this.x - 1);
  }

  public String toString() {
    return String.valueOf(this.x);
  }
}
```

(a) Consider the following method `foo`.

Foo Method

```
static A foo(A a) {
  return a.incr().decr();
}
```

(i) Write the asynchronous version of the method `foo` (called `fooAsync`) so that:

- it returns a `CompletableFuture<A>`, and
- the body of the method is executed asynchronously, with both `incr` and `decr` running in the same thread.

(ii) Repeat the above, but now allow for the possibility that `incr` and `decr` run in two separate threads.

(b) Suppose we have another method `bar`.

Bar Method

```
static A bar(A a) {
  return a.incr();
}
```

We would like to invoke `bar` using `bar(foo(new A(0)))`. Write the asynchronous version of the method `bar` (call it `barAsync`) so that it returns `CompletableFuture<A>` and the body of the method is executed asynchronously.

What is the asynchronous equivalent of the invocation `bar(foo(new A(0)))`?

(c) Suppose we have another method `baz`.

Baz Method

```
static A baz(A a, int x) {
  if (x == 0) {
    return new A(0);
  }
  return a.incr().decr();
}
```

Write the asynchronous version of `baz` called `bazAsync` so that it returns `CompletableFuture<A>` and the body of the method is executed asynchronously.

(d) Consider the following method `qux`.

Qux Method

```
A qux(A a) {
  try {
    a = a.decr();
    a = a.decr();
  } catch (IllegalStateException e) {
    System.out.println("error");
    a = new A(10);
  }
  return a.incr();
}
```

Write the asynchronous version of `qux` called `quxAsync` so that it returns `CompletableFuture<A>` and the body of the method is executed asynchronously and the exception is handled properly.

(e) Consider the following code snippet.

Code snippet

```
fooAsync(new A(0));
barAsync(new A(0));
bazAsync(new A(0), 1);
System.out.println("done.");
```

Since the first three method calls are asynchronous, the string `"done."` is printed almost immediately, before the methods `fooAsync`, `barAsync` and `bazAsync` complete execution. Revise the code so that `"done."` is printed only after all three methods are complete, while allowing the possibility of `fooAsync`, `barAsync` and `bazAsync` running concurrently.

(f) Consider the following code snippet.

Code snippet

```
CompletableFuture<A> cf1 = fooAsync(new A(0));
CompletableFuture<A> cf2 = fooAsync(new A(1));
cf1.join();
System.out.println(cf2.join().incr());
```

The code waited for both calls to `fooAsync` to complete before incrementing and then printing the value inside `A`. The result from the second `fooAsync` is always incremented and printed. Revise the code so that it increments and prints the value inside `A` as soon as one of `cf1` or `cf2` completes, while allowing the possibility of the two `CompletableFuture`s running concurrently. The code should prints either 1 or 2, depending on which call completes first.

2. Consider the following methods where `slowTask` always take 1 second to complete.

Zero Method

```
public Integer zero() {
  slowTask(); // Takes 1 second
  return 0;
}
```

AddOne Method

```
public Integer addOne(Integer i) {
  slowTask(); // Takes 1 second
  return i + 1;
}
```

Sum Method

```
public Integer sum(Integer i, Integer j) {
  slowTask(); // Takes 1 second
  return i + j;
}
```

Further consider the following code snippets.

(i) **Code Snippet 1**

```
CompletableFuture<Integer> cf1 = CompletableFuture.supplyAsync(() -> zero());
CompletableFuture<Integer> cf2 = cf1.thenApplyAsync(x -> addOne(x));
CompletableFuture<Integer> cf3 = cf1.thenApplyAsync(x -> addOne(x));
cf2.thenCombineAsync(cf3, (x, y) -> sum(x, y)).join();
```

(ii) **Code Snippet 2**

```
CompletableFuture<Integer> cf1 = CompletableFuture.supplyAsync(() -> zero());
CompletableFuture<Integer> cf2 = cf1.thenApply(x -> addOne(x));
CompletableFuture<Integer> cf3 = cf1.thenApply(x -> addOne(x));
cf2.thenCombineAsync(cf3, (x, y) -> sum(x, y)).join();
```

(iii) **Code Snippet 3**

```
CompletableFuture<Integer> cf1 = CompletableFuture.supplyAsync(() -> zero());
CompletableFuture<Integer> cf2 = cf1.thenApplyAsync(x -> addOne(x));
CompletableFuture<Integer> cf3 = cf1.thenApply(x -> addOne(x));
cf2.thenCombine(cf3, (x, y) -> sum(x, y)).join();
```

(iv) **Code Snippet 4**

```
CompletableFuture<Integer> cf1 = CompletableFuture.supplyAsync(() -> zero());
CompletableFuture<Integer> cf2 = cf1.thenApply(x -> addOne(x));
CompletableFuture<Integer> cf3 = cf1.thenApplyAsync(x -> addOne(x));
cf2.thenCombine(cf3, (x, y) -> sum(x, y)).join();
```

Compare the time taken to run the code excerpts above, focusing only on the delay caused by `slowTask()` and ignore any other threading or computational overhead. We assume that an infinite number of worker threads are available, and the system can run at maximum concurrency.

How long does each code snippet take to run? **Explain your answers**.

Exercise

The following questions are to be discussed on Ed as no answer will be given. ***Use only concepts that you have learnt****. You are expected to attempt them before the next lecture as the concepts may be used and/or mentioned in the lecture. The concept may also be tested for assessment.*

1. The idea of `fork()` and `join()` appears in multiple classes with different context and behavior. In some classes, `fork()` has different name to give more explicit control to the programmer. For each of the following classes, describe the name used for the idea of `fork()` and explain the behavior of `join()` in the context of that class.
   (a) `Thread` [2]
   (b) `CompletableFuture<T>` [1]
   (c) `RecursiveTask<T>` [3]
2. Consider the following simple method `sum`.

   Sum Method

   ```
   int sum(int lo, int hi, int[] arr) {
     int res = 0;
     for (int i = lo; i < hi; i++) {
       res += arr[i];
     }
     return res;
   }
   ```

   (a) Rewrite the code into a concurrent execution using `CompletableFuture` as follows.
   - If the size of `arr` is smaller than `THRESHOLD` (to be determined manually), then execute a sequential sum.
   - Otherwise, create two `CompletableFuture`s.
     - The first `CompletableFuture` is tasked with summing only the first half of `arr`.
     - The second `CompletableFuture` is tasked with summing only the second half of `arr`.
     - If the halves are still larger than `THRESHOLD`, recursively split the task smaller.

   (b) Fix the `THRESHOLD` and modify the size of `arr` in the input. Find the **critical size** `s` such that the execution for size `s` is **efficient** while the execution of size `s + 1` is **inefficient**.
   (c) Explain the behavior above. Suggest a way to prevent such inefficiency.

Bibliography

[1] Oracle, “Java 25 API: Future.” [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/concurrent/CompletableFuture.html

[2] Oracle, “Java 25 API: Thread.” [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/Thread.html

[3] Oracle, “Java 25 API: RecursiveTask.” [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/concurrent/RecursiveTask.html