**CS2030S RECITATION 11**

**Q1**

Parallel: 1 main and may sub tasks and all these sub takss run at the same time to achieve the main

Async: run something and wait for it to complete while you go and do something else

If you don’t use completablefuture: gotta do busywait, which is very painful because you gotta take note of callback() / time the thing yourself

CompletableFuture => functor and a monad, wraps something of type T, then use the methods listed

Analogy to car usage: you dk how to repair/how it works, but you can drive the car.

RunAsync=> Runnable: () -> returns nothing

supplyAsync=> Supplier: () -> returns a U => java immediate forks a new thread. It returns type U stored in CompletableFuture

thenApply() => equivalent to map. But uses the same thread for the previous one

thenApplyAsync => yet another thread created for the new return type from the prev line (potentially, though you will never know if Java wants to o it or not)

Optional => handle null

Sandbox => handles exception

Stream => parallelism / iteration

CompletableFuture/Stage => Async prog

thenAccept does not wait for completion

if you do a get() / join at the end, it will end up blocking so later code will not run first

join() returns the result… jshell quicks attempts to print

CompletableStage is the interface, CompletableFuture is the class that implements the Stage.

1. Unit is in milliseconds in Thread.sleep(1000); foo() blocks the execution of subsequent code
2. CompletionStage and Future are synnonamous. thenCompose is like flatMap

So the difference between supplyAsync and completedFuture is that supplyAync spawns a new thread but CompletedFuture is just wrapping it.

1. Ss
2. CompletableFuture.allOf( foo(new A()), bar(new A()), baz(new A(), 1)).join();

System.out.println(“Done”);

Different from anyOf => when ANY of them are completed, with the result being the earliest thread completed.

1. Exception thrown. Gotta use CompletedStage.handle

Var me = CompletableFuture.supplyAsync(() -> new A().decr().decr())

.handle( (result,exception) -> {

if (result == null) {

System.out.println(“ERROR” + ex);

Return new A();

}

Return result;

});

.handle provides a catch block in case any of the inside throws

Q2: thenCombine c.thenCombine(d). I is a bifunction

Non-blocking behaviour

Join to wait for stuff to complete.