static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier);  
static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier, Executor executor);  
static CompletableFuture<Void> runAsync(Runnable runnable);  
static CompletableFuture<Void> runAsync(Runnable runnable, Executor executor);  
  
  
  
  
<U> CompletableFuture<U> thenApply(Function<? super T,? extends U> fn);  
<U> CompletableFuture<U> thenApplyAsync(Function<? super T,? extends U> fn);  
<U> CompletableFuture<U> thenApplyAsync(Function<? super T,? extends U> fn, Executor executor);  
  
CompletableFuture<Void> thenAccept(Consumer<? super T> block);  
CompletableFuture<Void> thenRun(Runnable action);  
  
future.thenAcceptAsync(dbl -> log.debug("Result: {}", dbl), executor);  
  
CompletableFuture<String> safe =  
    future.exceptionally(ex -> "We have a problem: " + ex.getMessage());  
  
  
CompletableFuture<Integer> safe = future.handle((ok, ex) -> {  
    if (ok != null) {  
        return Integer.parseInt(ok);  
    } else {  
        log.warn("Problem", ex);  
        return -1;  
    }  
});  
  
<U> CompletableFuture<U> thenCompose(Function<? super T,CompletableFuture<U>> fn);  
  
  
<U,V> CompletableFuture<V> thenCombine(CompletableFuture<? extends U> other, BiFunction<? super T,? super U,? extends V> fn);  
  
CompletableFuture<Customer> customerFuture = loadCustomerDetails(123);  
CompletableFuture<Shop> shopFuture = closestShop();  
CompletableFuture<Route> routeFuture =  
    customerFuture.thenCombine(shopFuture, (cust, shop) -> findRoute(cust, shop));  
  
  
  
<U> CompletableFuture<Void> thenAcceptBoth(CompletableFuture<? extends U> other, BiConsumer<? super T,? super U> block)  
CompletableFuture<Void> runAfterBoth(CompletableFuture<?> other, Runnable action);  
  
  
customerFuture.thenAcceptBoth(shopFuture, (cust, shop) -> {  
    final Route route = findRoute(cust, shop);  
    //refresh GUI with route  
});