CMSC 433 – Programming Language Technologies and Paradigms

Futures Design Recipe

Recall First Design Recipe

- Break work into chunks
- Process chunks in a single thread
- Identify potential thread safety issues and devise solutions
- Transform single-threaded program to equivalent multi-threaded program

Implementing Asynchronous Calls

- Follow first design recipe
- Convert method calls into asynchronous calls using Futures
- Iterate over Futures to extract unit results
  - Combine unit results into final result

Running Example

- Method call to foo() and a later use of the result

```
T z = o.foo(x,y);
   ...
z.bar();
```

Implement Callable<T>

- Create a separate class that implements Callable<T>
- Constructor takes original function's args as parameters
  - Stores them as private fields
- The call() method executes the orig call
  - Returns the result

```
public class FooCall implements Callable<T> {
   private T1 x; private T2 y; private T3 o;
   public FooCall(T1 x, T2 y, T3 o) {
     this.x = x;  this.y = y;  this.o = o;
   }
   T call() { return o.foo(x,y); }
 }
```

Replace Invocation Sites

- Create an instance of this new class and pass it to an ExecutorService
  - Returns a Future<T> for a Callable<T>

```
ExecutorService executor = Executors.newFixedThreadPool(N);
Future<T> futureZ = executor.submit(new FooCall(x,y,o));
```

Replace Result Use Sites

- Wherever result is used in original program, replace that use with a call to the Future.get()
  - This call blocks until the result is available
  - Alternatively, store result of Future.get() in z prior to the first use of z

futureZ.get().bar()