Joshua Catoe

CSCI 350

Reflection Assignment - Practice Asynchronous Programming

Part 2. Practicing Asynchronous Programming

Read and study the code at <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/index>

to learn about asynchronous programming. After you’re done, complete the following questions: 1-2.

1. Rewrite the final (or semi-final) program to include grits on the breakfast menu. (You can either build on the code from the section: “Composition with Tasks” or the section: “Await tasks efficiently”) You’ll want a BoilGrits method that takes no parameters, and also a step for adding butter to the grits after the boiling is done. Embed the new program, including the additional code for grits, here. **Add comments explaining each line of code. Assume the reader doesn’t know the concepts of Task, async, and await, so be sure to comment those in detail.**

**See page 2.**

Read the article at <https://medium.com/@cummingsi1993/the-difference-between-asynchronous-and-parallel-6400729fa897>.

1. Based on the author’s understanding of async and parallel, at what point in the breakfast tutorial does the code make the biggest jump from just async to more parallel?

**The biggest jump from asynchronous to parallel in this tutorial comes under the “Composition with tasks” section. The** eggsTask**,** baconTask**, and** toastTask **are all put into their own variables, and the results of these tasks are put into the subsequent** eggs**,** bacon**, and** toast **variables. Doing this ensures that each task will run on one thread separate from the others.**

/\* async – Specifies that a method is asynchronous; required to use the await operator

\*

\* Task – Specifies that the Task class is being used; the Task class allows the use of Task objects, which hold any

\* sort of operation

\*

\* await – Halts execution until task is complete

\*/

static async Task Main(string[] args)

{

Coffee cup = PourCoffee(); // Create new coffee object and pour it into cup

Console.WriteLine("coffee is ready"); // Print text to console

var eggsTask = FryEggsAsync(2); // Create new task eggsTask

var baconTask = FryBaconAsync(3); // Create new task baconTask

var toastTask = MakeToastWithButterAndJamAsync(2); // Create new task toastTask

var gritsTask = BoilGritsAsync(); // Create new task gritsTask

var allTasks = new List<Task>{eggsTask, baconTask, toastTask, gritsTask}; // Put all tasks in list of task objects

// Process this loop while any of the tasks in allTasks are not completed

while (allTasks.Any())

{

Task finished = await Task.WhenAny(allTasks); // Hold the result of any of the tasks in allTasks

// Process finished tasks

if (finished == eggsTask)

{

Console.WriteLine("eggs are ready"); // If eggsTask is finished, print text to console

}

else if (finished == baconTask)

{

Console.WriteLine("bacon is ready"); // If baconTask is finished, print text to console

}

else if (finished == toastTask)

{

Console.WriteLine("toast is ready"); // If toastTask is finished, print text to console

}

else if (finished == gritsTask)

{

Console.WriteLine(“grits are ready”); // If gritsTask is finished, print text to console

}

allTasks.Remove(finished); // Remove task from allTasks when it has completed

}

Console.WriteLine("Breakfast is ready!"); // Print text to console

/\* Makes toast and applies butter and jam.

\*

\* Parameters: int number – number of pieces of toast

\*

\*/ Returns: toast task

async Task<Toast> MakeToastWithButterAndJamAsync(int number)

{

var toast = await ToastBreadAsync(number); // Toast bread with ToastBreadAsync method and receive toast object

ApplyButter(toast); // Apply butter to toast object

ApplyJam(toast); // Apply jam to toast object

return toast; // Return toast task

}

/\* Boils grits and adds butter asynchronously.

\*

\* Parameters – N/A

\*

\* Returns: grits task

\*/

async Task<Grits> BoilGritsAsync()

{

var grits = await boilGrits(); // Recieve grits object from boilGrits()

ApplyButter(grits); // Add butter to grits object

return grits; // Return grits task to calling function

}

/\* Boils grits.

\*

\* Parameters – N/A

\*

\* Returns: grits object

\*/

Grits boilGrits()

{

Grits grits = new Grits; // Create new grits object for boiling

Boil(grits); // Use Boil method to boil grits

return grits; // Return boiled grits object to calling method

}

}