Task one:
There are 4 actors.
Actor Alice makes objects called “wheat” and sends them to charlie.
When Alice receives an actor object as a message, it creates newly made wheat and sends it to the actor object from the message earlier, using .tell(). Then it messages itself with the received actor object again, thus creating an endless loop. In this case, the actor mentioned in the original message is Charlie, thus Alice endlessly sends wheat to Charlie.

Actor Bob makes objects called “sugar” and sends them to charlie.
When Bob receives an actor object as a message, it creates newly made sugar and sends it to the actor object from the message earlier, using .tell(). Then it messages itself with the received actor object again, thus creating an endless loop.In this case, the actor mentioned in the original message is Charlie, thus Bob endlessly sends sugar to Charlie.

Actor Charlie receives sugar, wheat and sends cakes to Tim
When Charlie receives sugar, Charlie checks if he has wheat stored in his List of wheat. If it doesn’t, he stores the sugar in his List of sugar. If it does have wheat, it will make cake using both the sugar and wheat, and send it to Tim.

Actor Tim receives the cakes.
Tim is initialized with a hunger variable. The number ticks downwards when it receives a cake. Once the number reaches zero, Tim sends a message of a “Gift” object back to the Cakes main() thread object.

Cakes is the main class with the main method that initializes the entire process.
First, it creates the ActorSystem through AkkaConfig with the configuration details; configuration details include the System Name, ‘this’ Port Number, and the ipv4 addresses of the other actors.
Then, it creates all four actors and begins the endless loop by sending Alice and Bob a message of (Actor Charlie) with (Actor Tim) as the sender. This ensures that Alice and Bob will send their products to Charlie, and Charlie will ‘return’ cakes to Tim.
Lastly, It asks Tim for a gift, submits the asking task to a Completable future, and waits for Tim to respond using a .join() method. When Tim’s hunger reaches zero, he will send the gift as requested back to this CompletableFuture as a return value.

When the gift is received, all actors are killed using a PoisonPill, and the system is terminated.

Task Three:
Testing was done by printing the difference between the timestamp before beginning the program, and after. The result printed is the exact number of milliseconds elapsed.

The option to run on additional machines was unavailable. Everything was tested on the singular machine.
Therefore, running the program with four Bob’s yielded results no different to that of one Bob.