Validating your simulation…

**SimulationEvent Class**

The `SimulationEvent` class acts as a factory.

Each object has the following fields, though they are set to null or 0 if they do not apply to a particular event type:

- `Cook cook;`
- `Customer customer;`
- `Machine machine;`
- `Food food;`
- `List<Food> orderFood;`
- `int orderNumber;`
- `int[] simParams;`

A valid list of events should start with a `SimulationStarting` event and end with a `SimulationEnded` event.

Within the `SimulationStarting` event, the `simParams` instance variable will point to an array that contains values for the number of customers, the number of cooks, the number of tables, and the capacity of the machines.
**A simple test...**

Once you have the first event in the list, and have confirmed that it is in fact a `SimulationStarting` event, you can extract the parameters that were used for that particular simulation run.

Once you have these, you can confirm that the rest of the details of the events list don’t violate those parameters.

For example, you can count how many customers appear within an events list by creating a list of strings (a `Vector` might be useful for example) and build a list of customer names by iterating over the events list. If an event contains a `Customer` reference, you can extract the name of the customer, see if it already appears in the list, and add it if it doesn’t. Then you just need to make sure the size of the list matches the number of customers the coffee shop was meant to serve.

**A little more involved test...**

A slightly more involved test, but still a good one to start with, would be determining what the maximum number of tables in use during the simulation was, and making sure that didn’t exceed the simulation parameter for that.

Again you could iterate through the events, but this time looking for the `CustomerEnteredCoffeeShop` and `CustomerLeavingCoffeeShop` events. You can increase and decrease the current occupancy level of the coffee shop as you traverse the list of events. You can keep a “high water mark” for this value. You can make sure this “high water mark” never exceeds the simulation limit. You can make sure that at the end, there are no customers left in the coffee shop.

Think about similar tests for things like the capacity of each of the machines that is making the food.
Testing for very specific errors...
Consider specific types of errors that could occur.
- What if someone receives an order before the order is actually completed?
- What if someone leaves the coffee shop without ever receiving their order?
- What if a customer places more than one order?
- etc.