**Pizza Factory Solution**

**How the application works:**

There are three main classes in this application:

* Program / Main (run when the application is started)
* Factory (handles pizza creation)
* Pizza (specifies details about a pizza)

The main function / class reads configuration information from a file labelled “config.txt”, and stores the various configuration options for the pizza-factory such as:

* Base cooking time
* Interval between cooking pizzas
* Naming convention for files storing pizza description
* Number of pizzas the factory should produce

The main function also calls two static methods in the Factory, which read the ‘toppings.txt’ and ‘bases.txt’ files, storing their contents for use when randomly generating pizzas.

Next the factory class is initialized, followed by calling its ‘start()’ function.   
This generates the specified number of pizzas, then cooks them, storing a description for each pizza generated in a text file (following the naming convention specified in the config file).

Once done cooking all pizzas, it shuts down.

**Design Choices:**

I chose to have the toppings and bases stored in text files so that they were configurable to run the application.

The results for all the files read-in to the application are stored in Dictionaries.  
In the config file, the option name is used as a key and the value after the ‘=’ as the value. In this specific instance, since option types change, the values are strings – but are converted to integers as needed.

The topping and base dictionaries store int and double values respectively, with the topping value calculated as it is read-in, avoiding having to re-count the number of characters each time this value is needed. Using dictionaries means I can easily call an option / base / topping by name, and get its associated value as needed.

The topping and base dictionaries (and their file-read methods) are static, since the same files are read, regardless of factory instance – so I wanted to avoid spending time reading the files in a situation where multiple factories were created.

If the application were in charge of controlling multiple factories, each with their own configurations this could be easily changed.

Using this method of text-files and dictionaries means that the number of application options can be increased, as can the available toppings and bases for pizzas.

Another useful feature of this application is that the Factory class has separate methods for creating and cooking pizzas, meaning it could easily be adapted to multi-threading to simulate multiple pizza makers / ovens working in parallel.