Food factory exercise

We have to provide a solution for a factory that processes food products in different assembly lines. We need to automate the cooking portion of the assembly line for the products. The factory produces various food products, which are created in assembly lines. There is a common part to many of them, and that is the cooking stages for which some ovens are used. The process to cook the different products (in the “cooking stage”) in a given point of the assembly line involves getting the products from the line, to put them in the oven for a specific amount of time, in the order they arrive. An intermediate store is used for the products that arrive, if there is no space left in the oven, which has a finite size. If there is no more room in the ovens or the stores when extracting it from the assembly line, the originating assembly line halts. After each product is cooked, we have to extract it and return it to the originating line (this is because multiple lines arrive at this automated stage).
We have to develop an application that controls the cooking stage of the factory.
Considerations:

- We have N Stores and M Ovens to process the products
- There is no particular restriction of the oven to use or the store to put the products in (e.g.: different products can share the same store/oven at the same time)
- All the processing from the different lines should respect FIFO ordering for the cooking to simplify the solution. That is, the order in which the elements are picked from a line, should be the same after the cooking stages, when these are put back.
- There is no control over the rate at which the lines will make a product available to be taken.

The following interfaces have to be used (but do not need to be implemented):

```
/**
* Implementations of this class should take care of overriding the necessary methods of the Object class to allow
* for the use of Collections in the different implementations of Oven and Store.
* This interface is not required to be implemented for this exercise.
*/
public interface Product {

   /**
    * The size that this product physically occupies in cm2
    * @return
    */
   double size();

   /**
    * This is the duration that this product should be cooked for.
    */
   Duration cookTime();

}
```

```java
/**
* This interface represents the Oven that cooks the products in the different assembly lines
*/
public interface Oven {
  /**
   * This returns the size of the oven in cm2. As a simplification of the problem, assume that the
   * sizes of the products can be summed, and that value should not exceed the size of the oven.
Otherwise an
   * exception is thrown if adding a product.
   * @return
   */
  double size();

  /**
   * Puts a product in the oven to be cooked. The oven can be functioning at the time the product is put
in.
   * @param product The product to put in the oven
   * @throws CapacityExceededException if the oven capacity is exceeded.
   */
  void put(Product product) throws CapacityExceededException;

  /**
   * Take the specified Product out of the oven. The oven can be functioning at the time the product is
taken out.
   * @param product
   */
  void take(Product product);

  /**
   * Turns on the Oven. If the oven was turned on with a duration, the duration is ignored.
   */
  void turnOn();

  /**
   * Turn on the Oven for the specified duration. If the oven is turned on, it updates the duration.
   * @param duration the duration to mantain the oven before turning it off.
   */
  void turnOn(Duration duration);

  /**
   * Turn off the Oven immediately, even if it was turned on with a duration which will be ignored.
   */
  void turnOff();

}
```

```java
/**
* The store where to put the products if the oven is not avialable. This class is thread safe.
*/
public interface Store {

  /**
   * Put a product in this store, if there is no space left in the store, it will block
   * until enough space frees up. This operation will put the products in FIFO order
   * @param product The Product to put in this Store
   */
  void put(Product product);

  /**
   * Take the next element that has to be processed respecting FIFO
   * @return
   */
  Product take();

  /**
   * Take the specified Product from the Store
   * @param product
   */
  void take(Product product);
}
```

```java
* This represents an assembly line stage of the factory. Implementations of this class should be thread-safe
*/
public interface AssemblyLineStage {

  /**
   * Put the specified product to the assembly line to continue in the next stage.
   * @param product
   */
  void putAfter(Product product);

  /**
   * Takes the next product available from the assembly line.
   * @return
   */
  Product take();
}
```

Note:

- You are free to choose any data structures and Java elements you deem appropriate, and can submit any questions you have.