Food Factory Architectural Model

This is a general document that represents the whole design for this solution. This documentation is spartan and absolutely oriented to technical professionals. This document is a companion for the application and forms part of its basic documentation design.

DISCLAIMER: This document is not intended to be maintained in time, it just shows up the whole design and the coding that came as a result. The implementation of this could be found in the github repository: https://github.com/exemartinez/SimpleMultiThreadingExample

Food Factory Architectural Model
Version/Phase: 1.0 / Proposed
Ezequiel H. Martinez created on 6/14/2020. Last modified 6/14/2020

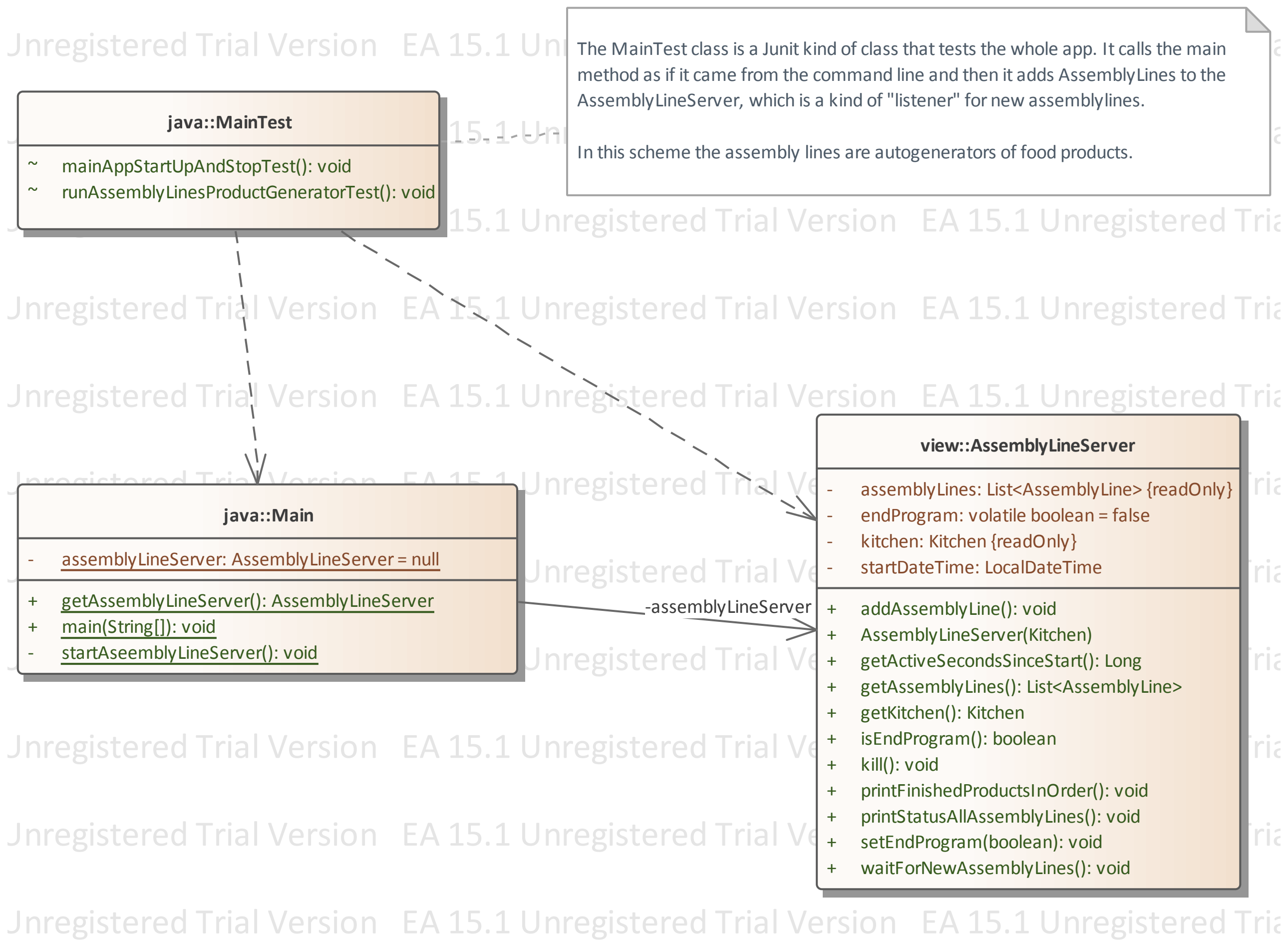
Food Factory Architectual Model 1
1. Class Structures 4
1.1 Execution and tests class structure diagram 4
1.1.1 Implemented Classes 5
Main 5
AssemblyLineServer 6
MainTest 8
1.2. Kitchen Class Structure diagram 10
Cooker 11
Kitchen 13
AssemblyLine 15
1.3 Main Class Structure diagram 19
KitchenBuilder 20
OvenBuilder 21
StoreBuilder 22
1.3 ‘Model’ Package Class Structure diagram 23
AssemblyLine 24
Food 27
OvenImpl 29
StoreImpl 30
AssemblyLineStage 31
Oven 32
Product 33
Store 34
2. Interactions 36
2.1. Kitchen Builder diagram 36
2.2. Assembly Line Manager diagram 36
2.3. Cooker Behaviour diagram 37
3. State Diagrams 40
3.1 State Diagrams diagram 40
4. Source Code Review 41

1. Class Structures

This is the whole set of classes that compose the system; explained and detailed in their relationship. Its intention is to show how the application architecture is intended.

1.1 Execution and tests class structure diagram

Here we expose how the main execution method is implemented: through automated test. We picked this methodology because it is imbued into almost all Java IDE's and the intended audience for this doc is purely technical.


The MainTest class is a Junit kind of class that tests the whole app. It calls the main method as if it came from the command line and then it adds AssemblyLines to the AssemblyLineServer, which is a kind of "listener" for new assemblylines.
In this scheme the assembly lines are autogenerators of food products.
java::MainTest
~ mainAppStartUpAndStopTest(): void
~ runAssemblyLinesProductGeneratorTest(): void
java::Main
- assemblyLineServer: AssemblyLineServer = null
+ getAssemblyLineServer(): AssemblyLineServer
+ main(String[]): void
- startAseemblyLineServer(): void
-assemblyLineServer
view::AssemblyLineServer
- assemblyLines: List<AssemblyLine> {readOnly}
- endProgram: volatile boolean = false
- kitchen: Kitchen {readOnly}
- startDateTime: LocalDateTime
+ addAssemblyLine(): void
+ AssemblyLineServer(Kitchen)
+ getActiveSecondsSinceStart(): Long
+ getAssemblyLines(): List<AssemblyLine>
+ getKitchen(): Kitchen
+ isEndProgram(): boolean
+ kill(): void
+ printFinishedProductsInOrder(): void
+ printStatusAllAssemblyLines(): void
+ setEndProgram(boolean): void
+ waitForNewAssemblyLines(): void
Unregistered Trial Version EA 15.1 Unregistered Trial Version

NOTE

The MainTest class is a Junit kind of class that tests the whole app. It calls the main method as if it came from the command line and then it adds AssemblyLines to the AssemblyLineServer, which is a kind of "listener" for new *AssemblyLines*.

In this scheme the assembly lines are *autogenerators* of food products.

1.1.1 Implemented Classes

Main

Class in package 'java'

A simulation of the following use case:

"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."

CONNECTORS

Dependency Source -> Destination
From: Main : Class, Public
To: KitchenBuilder : Class, Public

Dependency Source -> Destination
From: MainTest : Class, Public
To: Main : Class, Public

ATTRIBUTES

assemblyLineServer : AssemblyLineServer Private = null
[Is static True. Containment is Not Specified.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Main	Target: Private assemblyLineServer (Class) AssemblyLineServer

OPERATIONS

getAssemblyLineServer () : AssemblyLineServer Public
[Is static True. Is abstract False. Is return array False. Is query False. Is synchronized False.]

main (args : String[]) : void Public

This whole method, just builds the kitchen and starts to "hear" for requests. This stays looping. If this were an "actual" app, I'll put this hearing requests in a socket. Instead of it, we will work with a Junit that will hit some shared objects and notify it of a new assembly line.

[Is static True. Is abstract False. Is return array False. Is query False. Is synchronized False.]

startAseemblyLineServer () : void Private

Starts a looping thread that heards for request from different assembly lines.

[Is static True. Is abstract False. Is return array False. Is query False. Is synchronized False.]

AssemblyLineServer

Class in package 'view'

Receives all the requests from an external source. For this implementation, we will use a unit test. For future use, we will go for the implementation of a REST api (Throught Spring MVC) or hearing in a socket directly. Its major aim is to build the kitchen, start it up. And hear for new assembly lines to appear, dynamically over an external source. Every need assembly line will generate its own products.

CONNECTORS

Dependency Source -> Destination

From:	MainTest : Class, Public
To:	AssemblyLineServer : Class, Public

ATTRIBUTES

assemblyLines : List<AssemblyLine> Private Const

This is not how the request will be implemented in a real app!

[Is static True. Containment is Not Specified.]

endProgram : boolean Private = false

[Is static True. Containment is Not Specified.]

kitchen : Kitchen Private Const

[Is static True. Containment is Not Specified.]

startDateTime : LocalDateTime Private

[Is static True. Containment is Not Specified.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) AssemblyLineServer

Target: Private kitchen (Class) Kitchen

Association (direction: Source -> Destination)

Source: Public (Class) AssemblyLineServer

Target: Private assemblyLines (Class) AssemblyLine
Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Main

Target: Private assemblyLineServer (Class) AssemblyLineServer

OPERATIONS

addAssemblyLine () : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

AssemblyLineServer (kitchen : Kitchen) : Public

Defines the major controllers to operate the simulation. It starts the kitchen and defines the assembly lines.

Properties:
throws = KitchenRequiredException
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getActiveSecondsSinceStart () : Long Public

Just returns the amount of seconds since the current object started to hear requests.
@return
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getAssemblyLines () : List<AssemblyLine> Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getKitchen () : Kitchen Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

isEndProgram () : boolean Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

kill () : void Public

We kill the process in cold blood; losing state and data.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

printFinishedProductsInOrder () : void Public

This prints in the standard output the products already processed and finished. We use this only in the context of this simulation.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

printStatusAllAssemblyLines () : void Public

We go AssemblyLine by AssemblyLine Asking for the number of Elements in its "waiting" and "finished" lines.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

setEndProgram (endProgram : boolean) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

waitForNewAssemblyLines () : void Public

Loops for requests over the creation of new assembly lines (Threads), and starts all the simulation.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

MainTest

Class in package 'java'

Executes and test different the creation of new assembly lines and puts the simulation in motion.

CONNECTORS

Dependency Source -> Destination

From:	MainTest : Class, Public
To:	AssemblyLineServer : Class, Public

Dependency Source -> Destination

From:	MainTest : Class, Public
To:	Main : Class, Public

OPERATIONS

- mainAppStartUpAndStopTest () : void Package

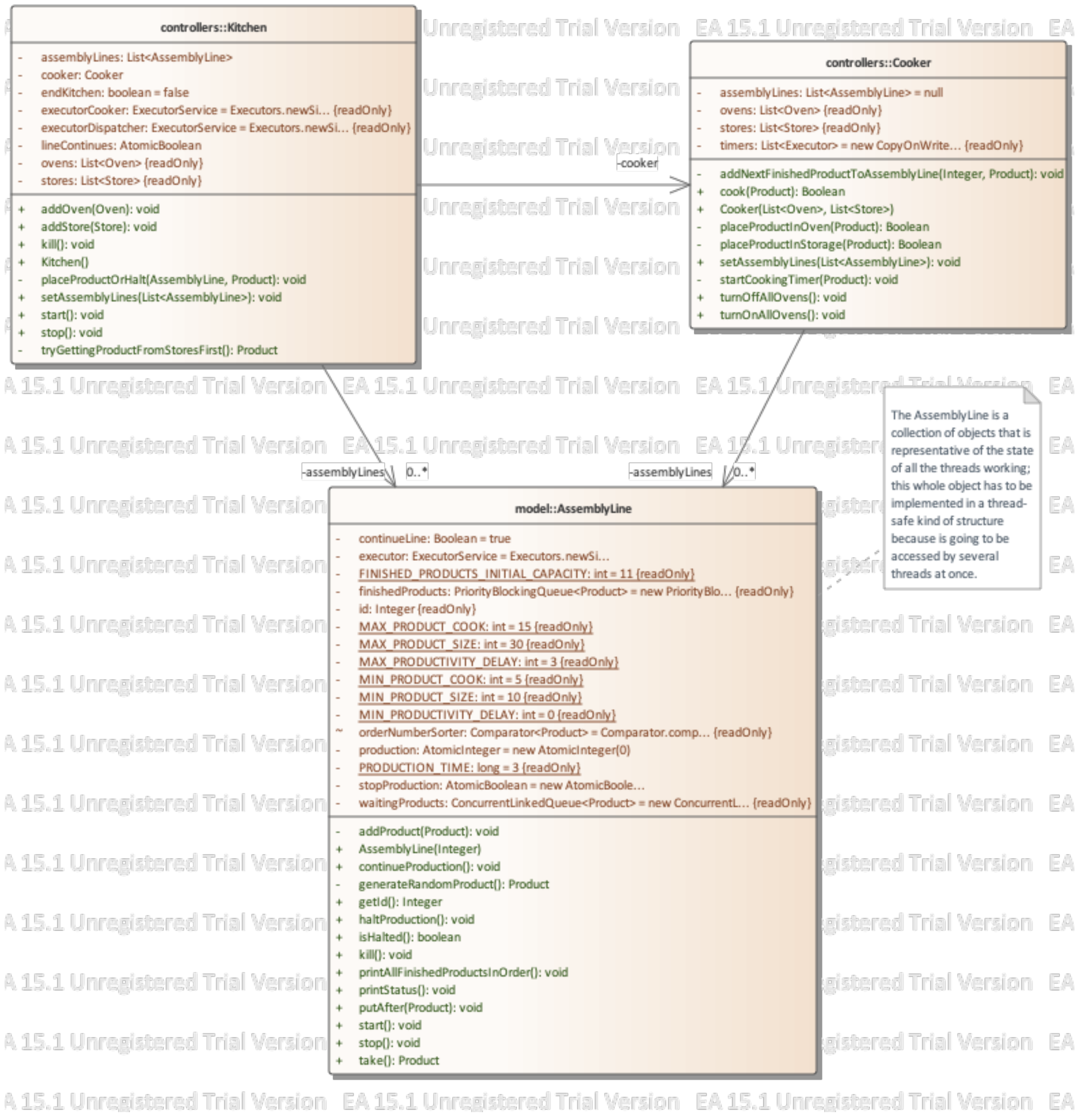
  Properties:
  annotations = @Test
  [Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

- runAssemblyLinesProductGeneratorTest () : void Package

  Properties:
  annotations = @Test
  [Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

1.2. Kitchen Class Structure diagram

This is the structure of the main functionality: the Kitchen and the tasks reserved to the Cooker.


controllers::Kitchen
- assemblyLines: List<AssemblyLine>
- cooker: Cooker
- endKitchen: boolean = false
- executorCooker: ExecutorService = Executors.newSi... {readOnly}
- executorDispatcher: ExecutorService = Executors.newSi... {readOnly}
- lineContinues: AtomicBoolean
- ovens: List<Oven> {readOnly}
- stores: List<Store> {readOnly}
+ addOven(Oven): void
+ addStore(Store): void
+ kill(): void
+ Kitchen()
- placeProductOrHalt(AssemblyLine, Product): void
+ setAssemblyLines(List<AssemblyLine>): void
+ start(): void
+ stop(): void
- tryGettingProductFromStoresFirst(): Product
-cooker
controllers::Cooker
- assemblyLines: List<AssemblyLine> = null
- ovens: List<Oven> {readOnly}
- stores: List<Store> {readOnly}
- timers: List<Executor> = new CopyOnWrite... {readOnly}
- addNextFinishedProductToAssemblyLine(Integer, Product): void
+ cook(Product): Boolean
+ Cooker(List<Oven>, List<Store>)
- placeProductInOven(Product): Boolean
- placeProductInStorage(Product): Boolean
+ setAssemblyLines(List<AssemblyLine>): void
- startCookingTimer(Product): void
+ turnOffAllOvens(): void
+ turnOnAllOvens(): void
-assemblyLines
0..*
-assemblyLines
0..*
The AssemblyLine is a collection of objects that is representative of the state of all the threads working; this whole object has to be implemented in a thread-safe kind of structure because is going to be accessed by several threads at once.
model::AssemblyLine
- continueLine: Boolean = true
- executor: ExecutorService = Executors.newSi...
- FINISHED_PRODUCTS_INITIAL_CAPACITY: int = 11 {readOnly}
- finishedProducts: PriorityBlockingQueue<Product> = new PriorityBlo... {readOnly}
- id: Integer {readOnly}
- MAX_PRODUCT_COOK: int = 15 {readOnly}
- MAX_PRODUCT_SIZE: int = 30 {readOnly}
- MAX_PRODUCTIVITY_DELAY: int = 3 {readOnly}
- MIN_PRODUCT_COOK: int = 5 {readOnly}
- MIN_PRODUCT_SIZE: int = 10 {readOnly}
- MIN_PRODUCTIVITY_DELAY: int = 0 {readOnly}
~ orderNumberSorter: Comparator<Product> = Comparator.comp... {readOnly}
- production: AtomicInteger = new AtomicInteger(0)
- PRODUCTION_TIME: long = 3 {readOnly}
- stopProduction: AtomicBoolean = new AtomicBoole...
- waitingProducts: ConcurrentLinkedQueue<Product> = new ConcurrentL... {readOnly}
- addProduct(Product): void
+ AssemblyLine(Integer)
+ continueProduction(): void
- generateRandomProduct(): Product
+ getId(): Integer
+ haltProduction(): void
+ isHalted(): boolean
+ kill(): void
+ printAllFinishedProductsInOrder(): void
+ printStatus(): void
+ putAfter(Product): void
+ start(): void
+ stop(): void
+ take(): Product

2. Kitchen Class Structure

NOTE

The AssemblyLine is a collection of objects that is representative of the state of all the threads working; this whole object has to be implemented in a thread-safe kind of structure because is going to be accessed by several threads at once.

Cooker

Class in package 'controllers'

Handles the thread of execution of the Kitchen. What needs to be cooked, in which order. It workd, more or less, like a "business controller" for the backend.

ATTRIBUTES

assemblyLines : List<AssemblyLine> Private = null
[Is static True. Containment is Not Specified.]

ovens : List<Oven> Private Const
[Is static True. Containment is Not Specified.]

stores : List<Store> Private Const
[Is static True. Containment is Not Specified.]

timers : List<Executor> Private Const = new CopyOnWriteArrayList<>()
[Is static True. Containment is Not Specified.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Cooker

Target: Private stores (Interface) Store
Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Cooker

Target: Private assemblyLines (Class) AssemblyLine
Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Cooker

Target: Private ovens (Interface) Oven
Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen

Target: Private cooker (Class) Cooker

OPERATIONS

addNextFinishedProductToAssemblyLine (idAssemblyLine : Integer , product : Product) : void Private

Adds one more product to the cache, for its given AssemblyLine
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

cook (product : Product) : Boolean Public Const

Here lies the main logic of how to put to cook a given product

Properties:
throws = InterruptedException
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Cooker (ovens : List<Oven> , stores : List<Store>) : Public

This constructor allows us to maintain a reference to the Kitchens ovens & stores for the cooker to manage them.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

placeProductInOven (product : Product) : Boolean Private

Tries and places a product in an Oven, if it is successful it starts a timer that simulates the cooking time.
@return
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

placeProductInStorage (product : Product) : Boolean Private

Tries to place the product in a Storage, if not, it returns false.
@return
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

setAssemblyLines (assemblyLines : List<AssemblyLine>) : void Public

We assign the assemblies lines to have access to the finishedProducts sorted queue.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

startCookingTimer (product : Product) : void Private

Start a thread with a timer to hold on it; when it finishes, we take the product out of the oven.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOffAllOvens () : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized True.]

turnOnAllOvens () : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized True.]

Kitchen

Class in package 'controllers'

This is the kitchen, it is composed of many ovens and stores. Assumption: the stores aren't indexed as one per OvenImpl; and the ovens doesn't has stores associated. Number of Ovens: N and Stores: M The kitchen starts the 'cooker' who is a virtual representation of a guy that performs the task of cooking and handling the products to the due assembly lines. There is a requirement: the products should be placed in the line, again in the same order in which they appeared. We solved this with an implementation of a "binary tree" that java has implemented as a "PriorityQueue"; this sort of data structure has the peculiarity of receiving items in any order and then retrieving them ordered. Neat!

Kitchen
Version 1.0 Phase 1.0 Proposed
Ezequiel H. Martinez created on 6/14/2020. Last modified 6/14/2020

ATTRIBUTES

assemblyLines : List<AssemblyLine> Private
[Is static True. Containment is Not Specified.]

cooker : Cooker Private
[Is static True. Containment is Not Specified.]

endKitchen : boolean Private = false
[Is static True. Containment is Not Specified.]

executorCooker : ExecutorService Private Const   = Executors.newSingleThreadExecutor()
[Is static True. Containment is Not Specified.]

executorDispatcher : ExecutorService Private Const   = Executors.newSingleThreadExecutor()
[Is static True. Containment is Not Specified.]

lineContinues : AtomicBoolean Private
[Is static True. Containment is Not Specified.]

ovens : List<Oven> Private Const
[Is static True. Containment is Not Specified.]

stores : List<Store> Private Const
[Is static True. Containment is Not Specified.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen	Target: Private ovens (Interface) Oven Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen	Target: Private stores (Interface) Store Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen	Target: Private cooker (Class) Cooker

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen	Target: Private assemblyLines (Class) AssemblyLine Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) AssemblyLineServer	Target: Private kitchen (Class) Kitchen

OPERATIONS

addOven (oven : Oven) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

addStore (store : Store) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

kill () : void Public

Brutal! but necessary option...
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Kitchen () : Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

placeProductOrHalt (assemblyLine : AssemblyLine , product : Product) : void Private

It searchs a place for the taken product, if there isn't any...the line halts.

Properties:
throws = InterruptedException
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

setAssemblyLines (assemblyLines : List<AssemblyLine>) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

start () : void Public

Starts the kitchen's "Cooker". A thread that monitors the assembly lines to take products to cook and retrieve them.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

stop () : void Public

Procedurally STOPS the whole set of processes as is.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

tryGettingProductFromStoresFirst () : Product Private

Before we go for the AssemblyLine, we go into the Stores and checkout for any remaining item to cook!
@return
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

AssemblyLine

Class in package 'model'

Represents the implementation of an hypothetical "Food Products Assembly Line" that is about to be simulated.

AssemblyLine
Version 1.0 Phase 1.0 Proposed
Ezequiel H. Martinez created on 6/14/2020. Last modified 6/14/2020

OUTGOING STRUCTURAL RELATIONSHIPS

Realization from AssemblyLine to AssemblyLineStage
[Direction is 'Source -> Destination'.]

ATTRIBUTES

continueLine : Boolean Private = true
[Is static True. Containment is Not Specified.]

◆ executor : ExecutorService Private = Executors.newSingleThreadExecutor()

[Is static True. Containment is Not Specified.]

◆ FINISHED_PRODUCTS_INITIAL_CAPACITY : int Private Const = 11

A prime number for good luck. :)

[Is static True. Containment is Not Specified.]

◆ finishedProducts : PriorityBlockingQueue<Product> Private Const = new PriorityBlockingQueue<>(FINISHED_PRODUCTS_INITIAL_CAPACITY,orderNumberSorter)

OUT queue

[Is static True. Containment is Not Specified.]

◆ id : Integer Private Const

[Is static True. Containment is Not Specified.]

◆ MAX_PRODUCT_COOK : int Private Const = 15

[Is static True. Containment is Not Specified.]

◆ MAX_PRODUCT_SIZE : int Private Const = 30

[Is static True. Containment is Not Specified.]

◆ MAX_PRODUCTIVITY_DELAY : int Private Const = 3

[Is static True. Containment is Not Specified.]

◆ MIN_PRODUCT_COOK : int Private Const = 5

[Is static True. Containment is Not Specified.]

◆ MIN_PRODUCT_SIZE : int Private Const = 10

DISCLAIMER: These values do not represent REAL cooking times neither food product sizes (just intented for this simulation use).

[Is static True. Containment is Not Specified.]

◆ MIN_PRODUCTIVITY_DELAY : int Private Const = 0

[Is static True. Containment is Not Specified.]

◆ orderNumberSorter : Comparator<Product> Package Const = Comparator.comparing(product -> ((Food)product).getOrderNumber())

[Is static True. Containment is Not Specified.]

production : AtomicInteger Private = new AtomicInteger(0)

[Is static True. Containment is Not Specified.]

PRODUCTION_TIME : long Private Const = 3

I like it every three seconds, a prime number takes the oddities to see some sunshine!

[Is static True. Containment is Not Specified.]

stopProduction : AtomicBoolean Private = new AtomicBoolean(false)

[Is static True. Containment is Not Specified.]

waitingProducts : ConcurrentLinkedQueue<Product> Private Const = new ConcurrentLinkedQueue<>()

IN queue

[Is static True. Containment is Not Specified.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Cooker	Target: Private assemblyLines (Class) AssemblyLine Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) AssemblyLineServer	Target: Private assemblyLines (Class) AssemblyLine Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen	Target: Private assemblyLines (Class) AssemblyLine Cardinality: [0..*]

OPERATIONS

addProduct (product : Product) : void Private

Adds the product to the "line" of products that need to be cooked by the Cooker on the multiple ovens.

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized True.]

AssemblyLine (id : Integer) : Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

continueProduction () : void Public

Wakes up the production.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

generateRandomProduct () : Product Private

I randomize the values that a new product might have, just before it enters the "input" line. I tries with minimal and maximum values but, ideally, those must be out of some sort of "setup scheme".
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getId () : Integer Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

haltProduction () : void Public

Stops the production, but the thread continues.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

isHalted () : boolean Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

kill () : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

printAllFinishedProductsInOrder () : void Public

Used to checkout if the products are in order
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

printStatus () : void Public

Prints the number of elements in each queue of the AssemblyLine. This should go into a file, a DB or a log; not to the standard output. However, for the intention of this exercise it will suffice.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

putAfter (product : Product) : void Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

start () : void Public

Initiates the generation of products and its due threads.

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

stop () : void Public

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

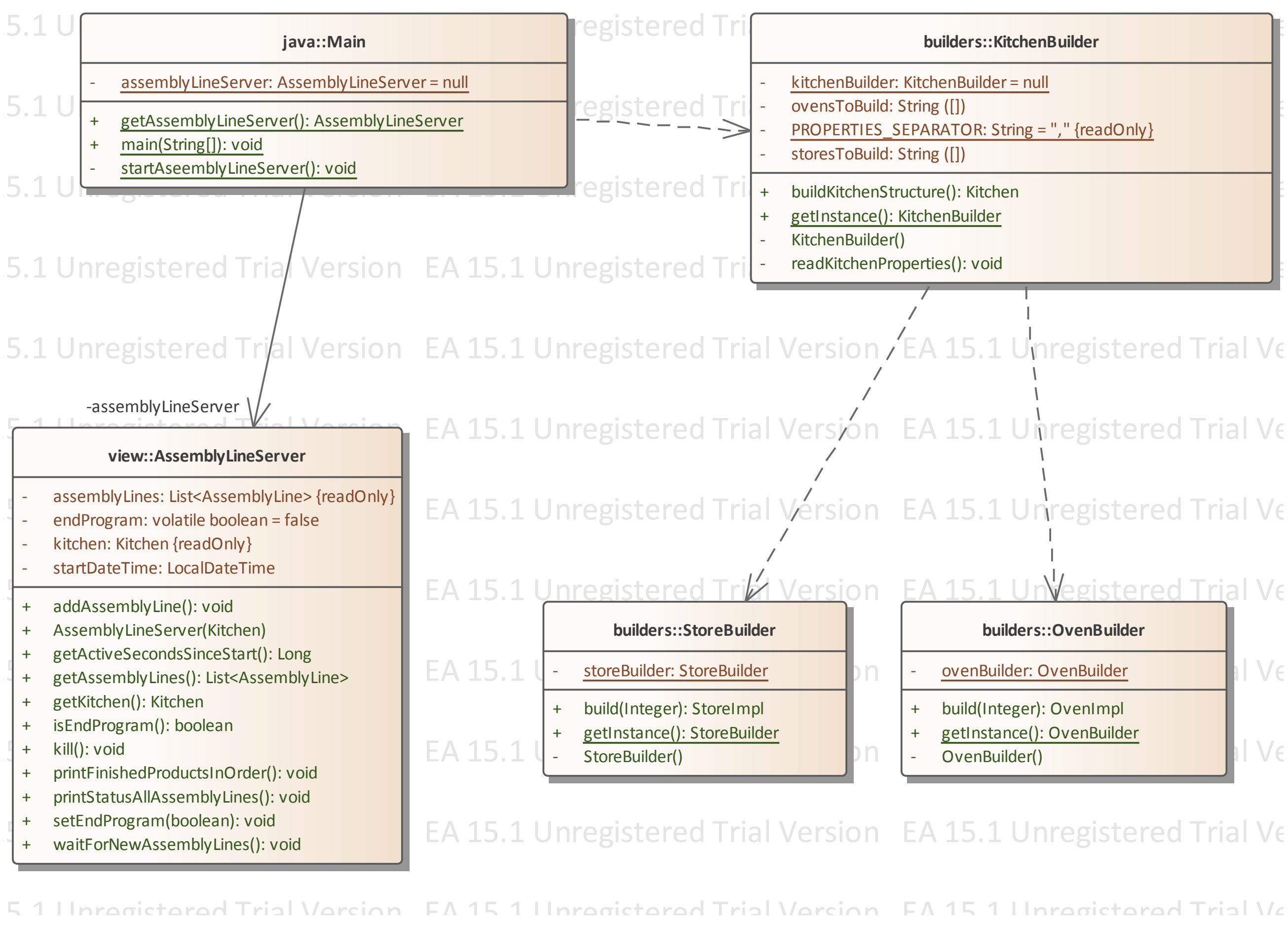
take () : Product Public

Properties:
annotations = @Override

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized True.]

1.3 Main Class Structure diagram

This represents the whole set of classes that runs as soon as the app is initialised. This end with a thread running indefinitely and hearing for new AssemblyLines to be called to be built.


java::Main
- assemblyLineServer: AssemblyLineServer = null
+ getAssemblyLineServer(): AssemblyLineServer
+ main(String[]): void
- startAseemblyLineServer(): void
builders::KitchenBuilder
- kitchenBuilder: KitchenBuilder = null
- ovensToBuild: String ([])
- PROPERTIES_SEPARATOR: String = "," {readOnly}
- storesToBuild: String ([])
+ buildKitchenStructure(): Kitchen
+ getInstance(): KitchenBuilder
- KitchenBuilder()
- readKitchenProperties(): void
-assemblyLineServer
view::AssemblyLineServer
- assemblyLines: List<AssemblyLine> {readOnly}
- endProgram: volatile boolean = false
- kitchen: Kitchen {readOnly}
- startDateTime: LocalDateTime
+ addAssemblyLine(): void
+ AssemblyLineServer(Kitchen)
+ getActiveSecondsSinceStart(): Long
+ getAssemblyLines(): List<AssemblyLine>
+ getKitchen(): Kitchen
+ isEndProgram(): boolean
+ kill(): void
+ printFinishedProductsInOrder(): void
+ printStatusAllAssemblyLines(): void
+ setEndProgram(boolean): void
+ waitForNewAssemblyLines(): void
builders::StoreBuilder
- storeBuilder: StoreBuilder
+ build(Integer): StoreImpl
+ getInstance(): StoreBuilder
- StoreBuilder()
builders::OvenBuilder
- ovenBuilder: OvenBuilder
+ build(Integer): OvenImpl
+ getInstance(): OvenBuilder
- OvenBuilder()
EA 15.1 Unregistered Trial Version

3. Main Class Structure

KitchenBuilder

Class in package 'builders'

Knows how to assemble a kitchen from a .properties file. This is a mixture between a Builder and a Factory pattern.

CONNECTORS

Dependency Source -> Destination
From: KitchenBuilder : Class, Public
To: StoreBuilder : Class, Public

Dependency Source -> Destination
From: KitchenBuilder : Class, Public
To: OvenBuilder : Class, Public

Dependency Source -> Destination
From: Main : Class, Public
To: KitchenBuilder : Class, Public

ATTRIBUTES

kitchenBuilder : KitchenBuilder Private = null
[Is static True. Containment is Not Specified.]

ovensToBuild : String Private
[Is static True. Containment is Not Specified.]

PROPERTIES_SEPARATOR : String Private Const = ","
[Is static True. Containment is Not Specified.]

storesToBuild : String Private
[Is static True. Containment is Not Specified.]

OPERATIONS

buildKitchenStructure () : Kitchen Public

Takes the due properties and builds up a kitchen as it's been requested in the properties files. NOTE: we could have added more properties to configure all the constants in the whole system; we avoided this approach for simplicity.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getInstance () : KitchenBuilder Public

Singleton kind of implementation as usually a builder/factory pattern could be implemented.
@return
[Is static True. Is abstract False. Is return array False. Is query False. Is synchronized False.]

KitchenBuilder () : Private

Builds itself from a .properties file
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

readKitchenProperties () : void Private

Loads up how to build the kitchen: how many Ovens and Stores, and their sizes.
@return
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

OvenBuilder

Class in package 'builders'

It is in charge of returning an "OvenImpl" kind of object. I do believe these objects could "evolve" during this development, so I try to encapsulate and detach the way in which we build them.

CONNECTORS

Dependency Source -> Destination
From: KitchenBuilder : Class, Public
To: OvenBuilder : Class, Public

ATTRIBUTES

ovenBuilder : OvenBuilder Private
[Is static True. Containment is Not Specified.]

OPERATIONS

build (size : Integer) : OvenImpl Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getInstance () : OvenBuilder Public
[Is static True. Is abstract False. Is return array False. Is query False. Is synchronized False.]

OvenBuilder () : Private
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

StoreBuilder

Class in package 'builders'

We detach the way we build the stores from the store itself. I do not know if we may want to change the very nature of what a Store 'is' later.

CONNECTORS

Dependency Source -> Destination

From:	KitchenBuilder : Class, Public
To:	StoreBuilder : Class, Public

ATTRIBUTES

storeBuilder : StoreBuilder Private
[Is static True. Containment is Not Specified.]

OPERATIONS

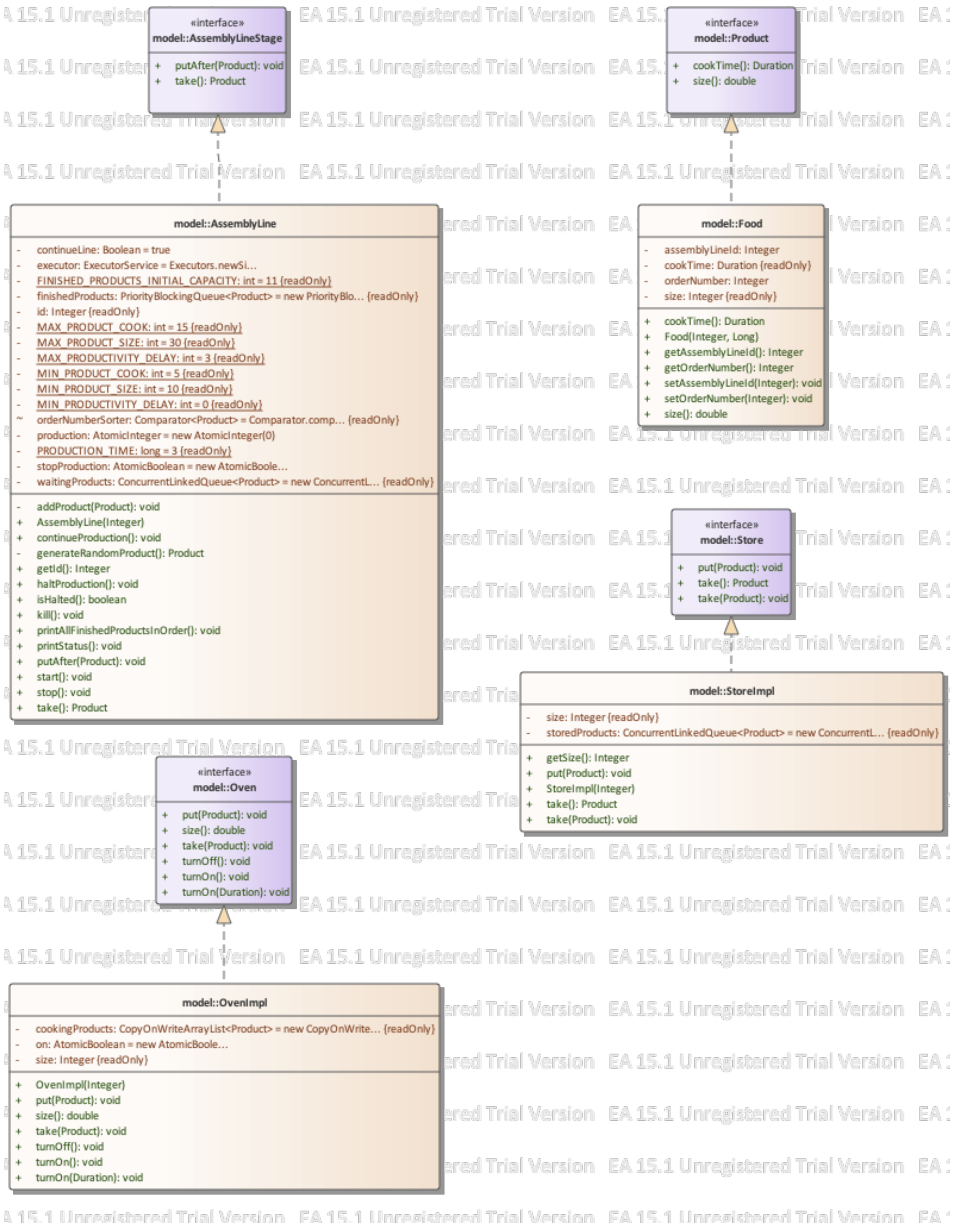
build (size : Integer) : StoreImpl Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getInstance () : StoreBuilder Public
[Is static True. Is abstract False. Is return array False. Is query False. Is synchronized False.]

StoreBuilder () : Private
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

1.3 ‘Model’ Package Class Structure diagram

Class diagram in package 'Class Structure'


«interface»
model::AssemblyLineStage
+ putAfter(Product): void
+ take(): Product
«interface»
model::Product
+ cookTime(): Duration
+ size(): double
model::AssemblyLine
- continueLine: Boolean = true
- executor: ExecutorService = Executors.newSi...
- FINISHED_PRODUCTS_INITIAL_CAPACITY: int = 11 {readOnly}
- finishedProducts: PriorityBlockingQueue<Product> = new PriorityBlo... {readOnly}
- id: Integer {readOnly}
- MAX_PRODUCT_COOK: int = 15 {readOnly}
- MAX_PRODUCT_SIZE: int = 30 {readOnly}
- MAX_PRODUCTIVITY_DELAY: int = 3 {readOnly}
- MIN_PRODUCT_COOK: int = 5 {readOnly}
- MIN_PRODUCT_SIZE: int = 10 {readOnly}
- MIN_PRODUCTIVITY_DELAY: int = 0 {readOnly}
~ orderNumberSorter: Comparator<Product> = Comparator.comp... {readOnly}
- production: AtomicInteger = new AtomicInteger(0)
- PRODUCTION_TIME: long = 3 {readOnly}
- stopProduction: AtomicBoolean = new AtomicBoole...
- waitingProducts: ConcurrentLinkedQueue<Product> = new ConcurrentL... {readOnly}
- addProduct(Product): void
+ AssemblyLine(Integer)
+ continueProduction(): void
- generateRandomProduct(): Product
+ getId(): Integer
+ haltProduction(): void
+ isHalted(): boolean
+ kill(): void
+ printAllFinishedProductsInOrder(): void
+ printStatus(): void
+ putAfter(Product): void
+ start(): void
+ stop(): void
+ take(): Product
model::Food
- assemblyLineId: Integer
- cookTime: Duration {readOnly}
- orderNumber: Integer
- size: Integer {readOnly}
+ cookTime(): Duration
+ Food(Integer, Long)
+ getAssemblyLineId(): Integer
+ getOrderNumber(): Integer
+ setAssemblyLineId(Integer): void
+ setOrderNumber(Integer): void
+ size(): double
«interface»
model::Store
+ put(Product): void
+ take(): Product
+ take(Product): void
model::StoreImpl
- size: Integer {readOnly}
- storedProducts: ConcurrentLinkedQueue<Product> = new ConcurrentL... {readOnly}
+ getSize(): Integer
+ put(Product): void
+ StoreImpl(Integer)
+ take(): Product
+ take(Product): void
«interface»
model::Oven
+ put(Product): void
+ size(): double
+ take(Product): void
+ turnOff(): void
+ turnOn(): void
+ turnOn(Duration): void
model::OvenImpl
- cookingProducts: CopyOnWriteArrayList<Product> = new CopyOnWrite... {readOnly}
- on: AtomicBoolean = new AtomicBoole...
- size: Integer {readOnly}
+ OvenImpl(Integer)
+ put(Product): void
+ size(): double
+ take(Product): void
+ turnOff(): void
+ turnOn(): void
+ turnOn(Duration): void
EA 15.1 Unregistered Trial Version

AssemblyLine

Class in package 'model'

Represents the implementation of an hypothetical "Food Products Assembly Line" that is about to be simulated.

AssemblyLine
Version 1.0 Phase 1.0 Proposed
Ezequiel H. Martinez created on 6/14/2020. Last modified 6/14/2020

OUTGOING STRUCTURAL RELATIONSHIPS

Realization from AssemblyLine to AssemblyLineStage
[Direction is 'Source -> Destination'.]

ATTRIBUTES

continueLine : Boolean Private = true
[Is static True. Containment is Not Specified.]

executor : ExecutorService Private = Executors.newSingleThreadExecutor()
[Is static True. Containment is Not Specified.]

FINISHED_PRODUCTS_INITIAL_CAPACITY : int Private Const = 11

A prime number for good luck. :)
[Is static True. Containment is Not Specified.]

finishedProducts : PriorityBlockingQueue<Product> Private Const = new PriorityBlockingQueue<>(FINISHED_PRODUCTS_INITIAL_CAPACITY,orderNumberSorter)

OUT queue
[Is static True. Containment is Not Specified.]

id : Integer Private Const
[Is static True. Containment is Not Specified.]

MAX_PRODUCT_COOK : int Private Const = 15
[Is static True. Containment is Not Specified.]

MAX_PRODUCT_SIZE : int Private Const = 30
[Is static True. Containment is Not Specified.]

MAX_PRODUCTIVITY_DELAY : int Private Const = 3

[Is static True. Containment is Not Specified.]

MIN_PRODUCT_COOK : int Private Const = 5

[Is static True. Containment is Not Specified.]

MIN_PRODUCT_SIZE : int Private Const = 10

DISCLAIMER: These values do not represent REAL cooking times neither food product sizes (just intented for this simulation use).

[Is static True. Containment is Not Specified.]

MIN_PRODUCTIVITY_DELAY : int Private Const = 0

[Is static True. Containment is Not Specified.]

orderNumberSorter : Comparator<Product> Package Const = Comparator.comparing(product -> ((Food)product).getOrderNumber())

[Is static True. Containment is Not Specified.]

production : AtomicInteger Private = new AtomicInteger(0)

[Is static True. Containment is Not Specified.]

PRODUCTION_TIME : long Private Const = 3

I like it every three seconds, a prime number takes the oddities to see some sunshine!

[Is static True. Containment is Not Specified.]

stopProduction : AtomicBoolean Private = new AtomicBoolean(false)

[Is static True. Containment is Not Specified.]

waitingProducts : ConcurrentLinkedQueue<Product> Private Const = new ConcurrentLinkedQueue<>()

IN queue

[Is static True. Containment is Not Specified.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Cooker	Target: Private assemblyLines (Class) AssemblyLine Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) AssemblyLineServer

Target: Private assemblyLines (Class) AssemblyLine
Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen

Target: Private assemblyLines (Class) AssemblyLine
Cardinality: [0..*]

OPERATIONS

addProduct (product : Product) : void Private

Adds the product to the "line" of products that need to be cooked by the Cooker on the multiple ovens.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized True.]

AssemblyLine (id : Integer) : Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

continueProduction () : void Public

Wakes up the production.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

generateRandomProduct () : Product Private

I randomize the values that a new product might have, just before it enters the "input" line. I tries with minimal and maximum values but, ideally, those must be out of some sort of "setup scheme".
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getId () : Integer Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

haltProduction () : void Public

Stops the production, but the thread continues.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

isHalted () : boolean Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

kill () : void Public

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

printAllFinishedProductsInOrder () : void Public

Used to checkout if the products are in order

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

printStatus () : void Public

Prints the number of elements in each queue of the AssemblyLine. This should go into a file, a DB or a log; not to the standard output. However, for the intention of this exercise it will suffice.

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

putAfter (product : Product) : void Public

Properties:
annotations = @Override

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

start () : void Public

Initiates the generation of products and its due threads.

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

stop () : void Public

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take () : Product Public

Properties:
annotations = @Override

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized True.]

Food

Class in package 'model'

A generic food. We can extend this class into hamburgers, fries or whatever.

OUTGOING STRUCTURAL RELATIONSHIPS

Realization from Food to Product

[Direction is 'Source -> Destination'.]

ATTRIBUTES

assemblyLineId : Integer Private
[Is static True. Containment is Not Specified.]

cookTime : Duration Private Const
[Is static True. Containment is Not Specified.]

orderNumber : Integer Private
[Is static True. Containment is Not Specified.]

size : Integer Private Const
[Is static True. Containment is Not Specified.]

OPERATIONS

cookTime () : Duration Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Food (size : Integer , cookTime : Long) : Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getAssemblyLineId () : Integer Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

getOrderNumber () : Integer Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

setAssemblyLineId (assemblyLineId : Integer) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

setOrderNumber (orderNumber : Integer) : void Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

size () : double Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

OvenImpl

Class in package 'model'

It cooks the food and handles its own storage of products

OUTGOING STRUCTURAL RELATIONSHIPS

Realization from OvenImpl to Oven
[Direction is 'Source -> Destination'.]

ATTRIBUTES

cookingProducts : CopyOnWriteArrayList<Product> Private Const = new CopyOnWriteArrayList<>()
[Is static True. Containment is Not Specified.]

on : AtomicBoolean Private = new AtomicBoolean(false)
[Is static True. Containment is Not Specified.]

size : Integer Private Const

TODO size was Double in the specification, refactor this.
[Is static True. Containment is Not Specified.]

OPERATIONS

OvenImpl (size : Integer) : Public
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

put (product : Product) : void Public

Properties:
annotations = @Override
throws = CapacityExceededException
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

size () : double Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take (product : Product) : void Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOff () : void Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOn () : void Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOn (duration : Duration) : void Public

Properties:
annotations = @Override
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

StoreImpl

Class in package 'model'

This holds the food, but doesn't cooks it.

OUTGOING STRUCTURAL RELATIONSHIPS

Realization from StoreImpl to Store
[Direction is 'Source -> Destination'.]

ATTRIBUTES

size : Integer Private Const

we will not provide a getter & setter for this

[Is static True. Containment is Not Specified.]

storedProducts : ConcurrentLinkedQueue<Product> Private Const = new ConcurrentLinkedQueue<>()

[Is static True. Containment is Not Specified.]

OPERATIONS

getSize () : Integer Public

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

put (product : Product) : void Public

Properties:
annotations = @Override

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

StoreImpl (size : Integer) : Public

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take () : Product Public

Properties:
annotations = @Override

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take (product : Product) : void Public

Properties:
annotations = @Override

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

AssemblyLineStage

Interface in package 'model'

This represents an assembly line stage of the factory. Implementations of this class should be thread-safe

INCOMING STRUCTURAL RELATIONSHIPS

Realization from AssemblyLine to AssemblyLineStage

[Direction is 'Source -> Destination'.]

OPERATIONS

putAfter (product : Product) : void Public

Put the specified product to the assembly line to continue in the next stage.

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take () : Product Public

Takes the next product available from the assembly line.
@return

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Oven

Interface in package 'model'

This interface represents the OvenImpl that cooks the products in the different assembly lines

INCOMING STRUCTURAL RELATIONSHIPS

Realization from OvenImpl to Oven

[Direction is 'Source -> Destination'.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen

Target: Private ovens (Interface) Oven
Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Cooker

Target: Private ovens (Interface) Oven
Cardinality: [0..*]

OPERATIONS

put (product : Product) : void Public

Puts a product in the oven to be cooked. The oven can be functioning at the time the product is put in.

Properties:
throws = CapacityExceededException
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

size () : double Public

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
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take (product : Product) : void Public

Take the specified Product out of the oven. The oven can be functioning at the time the product is taken out.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOff () : void Public

Turn off the OvenImpl immediately, even if it was turned on with a duration which will be ignored.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOn () : void Public

Turns on the OvenImpl. If the oven was turned on with a duration, the duration is ignored.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

turnOn (duration : Duration) : void Public

Turn on the OvenImpl for the specified duration. If the oven is turned on, it updates the duration.
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Product

Interface in package 'model'

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 OvenImpl and StoreImpl. This interface is not required to be implemented for this exercise.

INCOMING STRUCTURAL RELATIONSHIPS

Realization from Food to Product

[Direction is 'Source -> Destination'.]

OPERATIONS

cookTime () : Duration Public

This is the duration that this product should be cooked for.

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

size () : double Public

The size that this product physically occupies in cm2
@return

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

Store

Interface in package 'model'

The store where to put the products if the oven is not avialable. This class is thread safe.

INCOMING STRUCTURAL RELATIONSHIPS

Realization from StoreImpl to Store

[Direction is 'Source -> Destination'.]

ASSOCIATIONS

Association (direction: Source -> Destination)

Source: Public (Class) Kitchen	Target: Private stores (Interface) Store Cardinality: [0..*]

Association (direction: Source -> Destination)

Source: Public (Class) Cooker	Target: Private stores (Interface) Store Cardinality: [0..*]

OPERATIONS

put (product : Product) : void Public

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

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take () : Product Public

Take the next element that has to be processed respecting FIFO
@return

[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

take (product : Product) : void Public

Take the specified Product from the StoreImpl

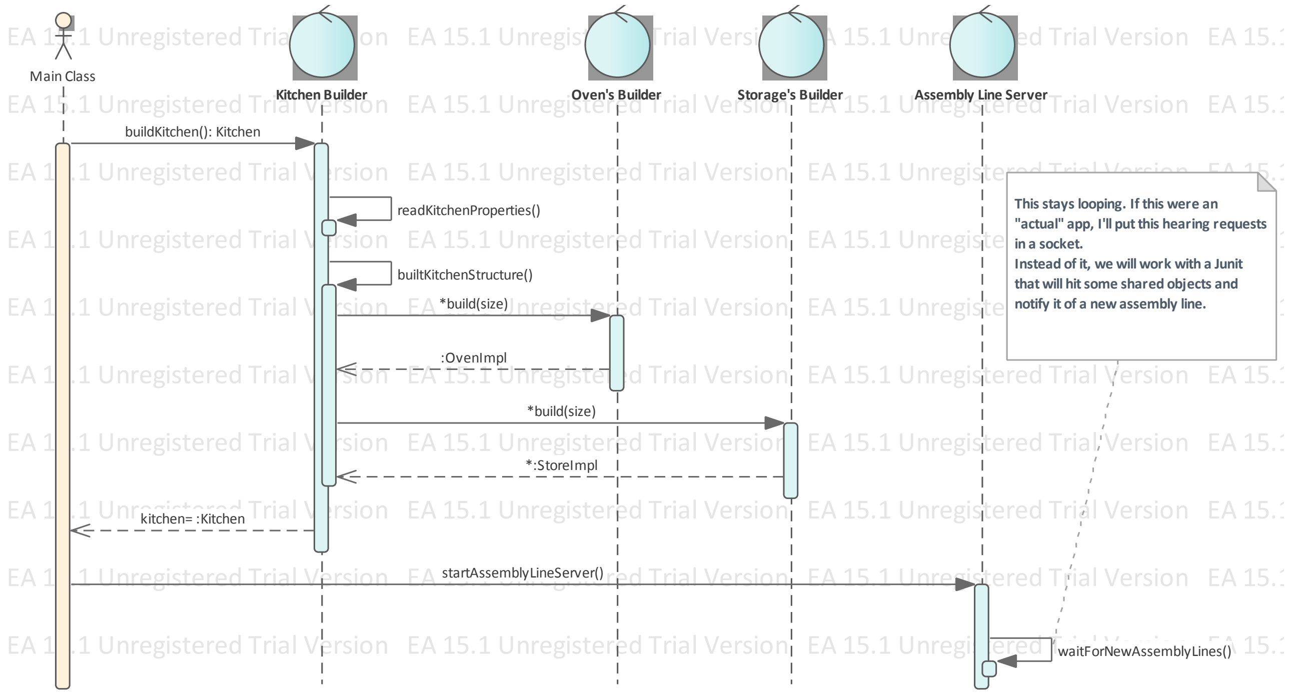
[Is static False. Is abstract False. Is return array False. Is query False. Is synchronized False.]

2. Interactions

Here we describe how the whole simulation app works. This is an ‘analysis’ stage design, that is why we found hypothetical classes and their desired role in the final app. We used this to model the whole functionality, minor changes may apply in the current/final implementation

2.1. Kitchen Builder diagram

This describes how the main class (inside the main thread) will build a Kitchen; interactions between different objects are being described.


Main Class
Kitchen Builder
Oven's Builder
Storage's Builder
Assembly Line Server
buildKitchen(): Kitchen
readKitchenProperties()
builtKitchenStructure()
*build(size)
:OvenImpl
*build(size)
*:StoreImpl
kitchen= :Kitchen
startAssemblyLineServer()
waitForNewAssemblyLines()
This stays looping. If this were an "actual" app, I'll put this hearing requests in a socket. Instead of it, we will work with a Junit that will hit some shared objects and notify it of a new assembly line.
EA 15.1 Unregistered Trial Version

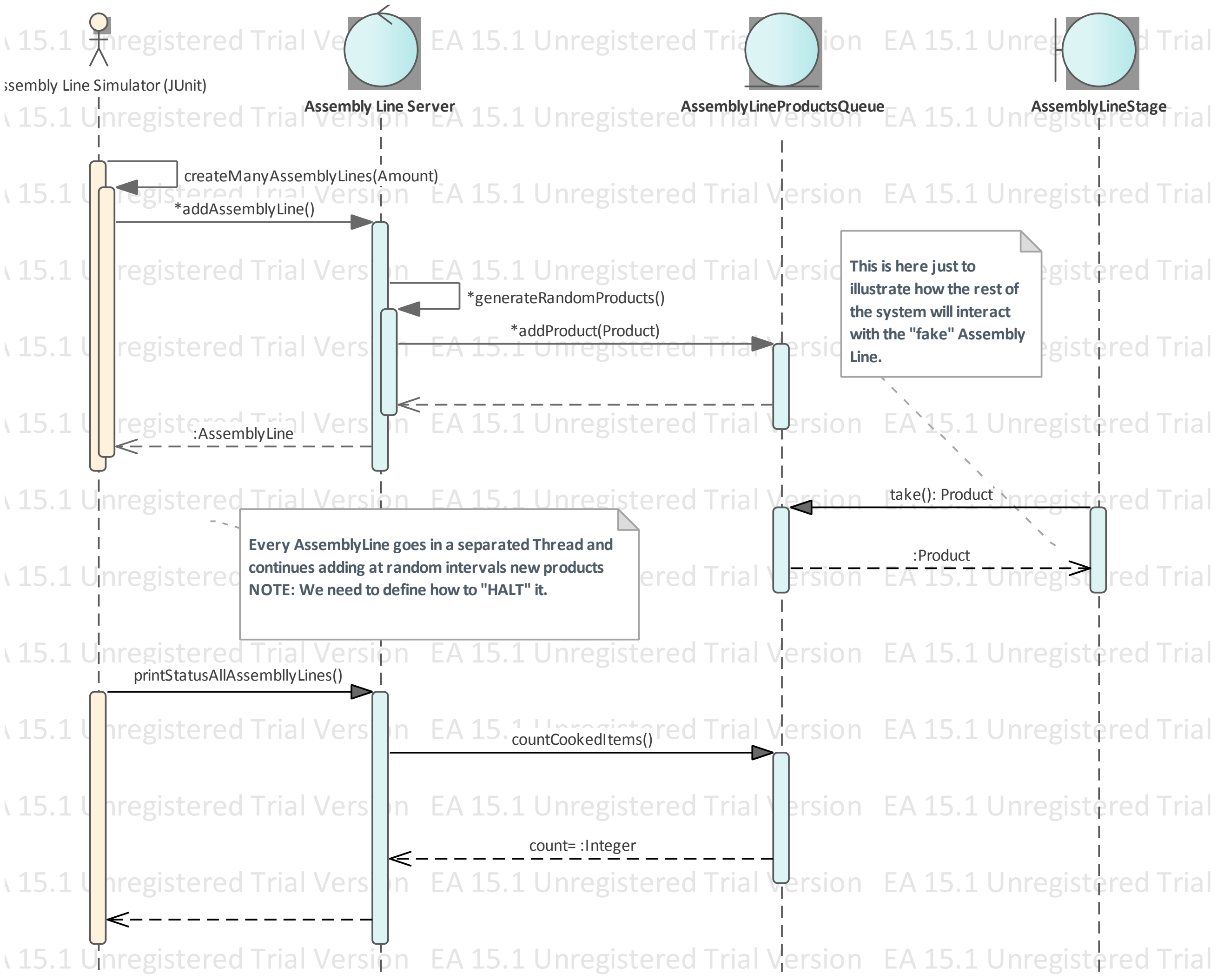
5. 1. Kitchen Builder

NOTE

This stays looping. If this were an "actual" app, I'll put this hearing requests in a socket.
Instead of it, we will work with a Junit that will hit some shared objects and notify it of a new assembly line appearance.

2.2. Assembly Line Manager diagram

Here we describe how the AssemblyLineServer will be called for adding a new assembly line over. The whole system looks after a continuos loop where the appearance of a new assembly line delivering products to the Kitchen might happen.


ssembly Line Simulator (JUnit)
Assembly Line Server
AssemblyLineProductsQueue
AssemblyLineStage
createManyAssemblyLines(Amount)
*addAssemblyLine()
*generateRandomProducts()
*addProduct(Product)
This is here just to illustrate how the rest of the system will interact with the "fake" Assembly Line.
:AssemblyLine
take(): Product
:Product
Every AssemblyLine goes in a separated Thread and continues adding at random intervals new products NOTE: We need to define how to "HALT" it.
printStatusAllAssemblllyLines()
countCookedItems()
count= :Integer
EA 15.1 Unregistered Trial Version

6. 2. Assembly Line Manager

NOTE

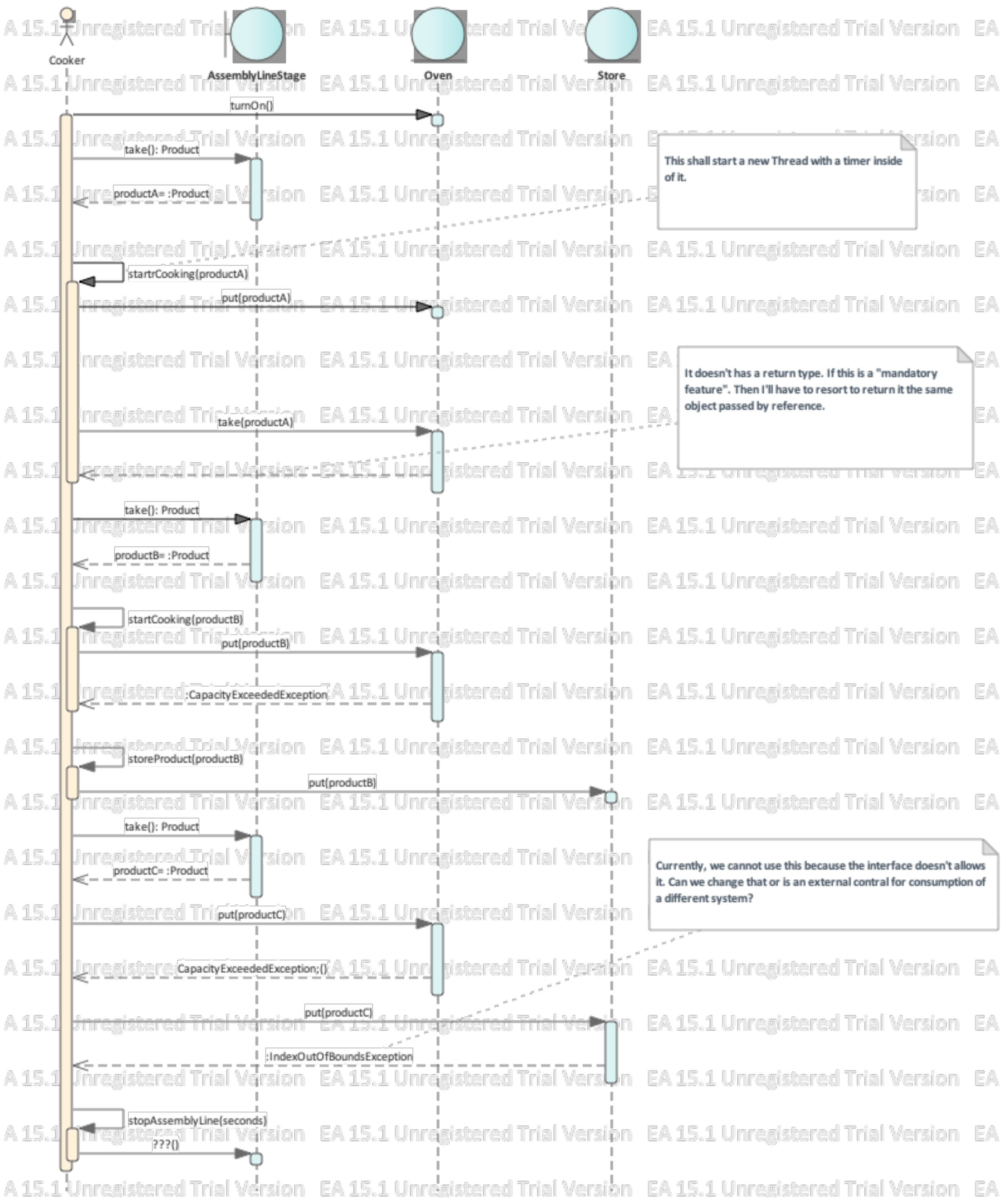
This is here just to illustrate how the rest of the system will interact with the "fake" Assembly Line.

NOTE

Every AssemblyLine goes in a separated Thread and continues adding at random intervals new products NOTE: We need to define how to "HALT" it.

2.3. Cooker Behaviour diagram

Here is where we describe the main behaviour: how a new product is introduced to the kitchen after it is graded by the cooker, and what happens then. The core application’s functionality is described here.


Cooker
AssemblyLineStage
Oven
Store
turnOn()
take(): Product
productA= :Product
This shall start a new Thread with a timer inside of it.
startrCooking(productA)
put(productA)
It doesn't has a return type. If this is a "mandatory feature". Then I'll have to resort to return it the same object passed by reference.
take(productA)
take(): Product
productB= :Product
startCooking(productB)
put(productB)
:CapacityExceededException
storeProduct(productB)
put(productB)
take(): Product
productC= :Product
Currently, we cannot use this because the interface doesn't allows it. Can we change that or is an external contral for consumption of a different system?
put(productC)
CapacityExceededException;()
put(productC)
:IndexOutOfBoundsException
stopAssemblyLine(seconds)
???()

7. 3. Cooker Behavior

NOTE

Currently, we cannot use this because the interface doesn't allows it. Can we change that or is an external contral for consumption of a different system?

NOTE

startCookingProduct() - This shall start a new Thread with a timer inside of it.

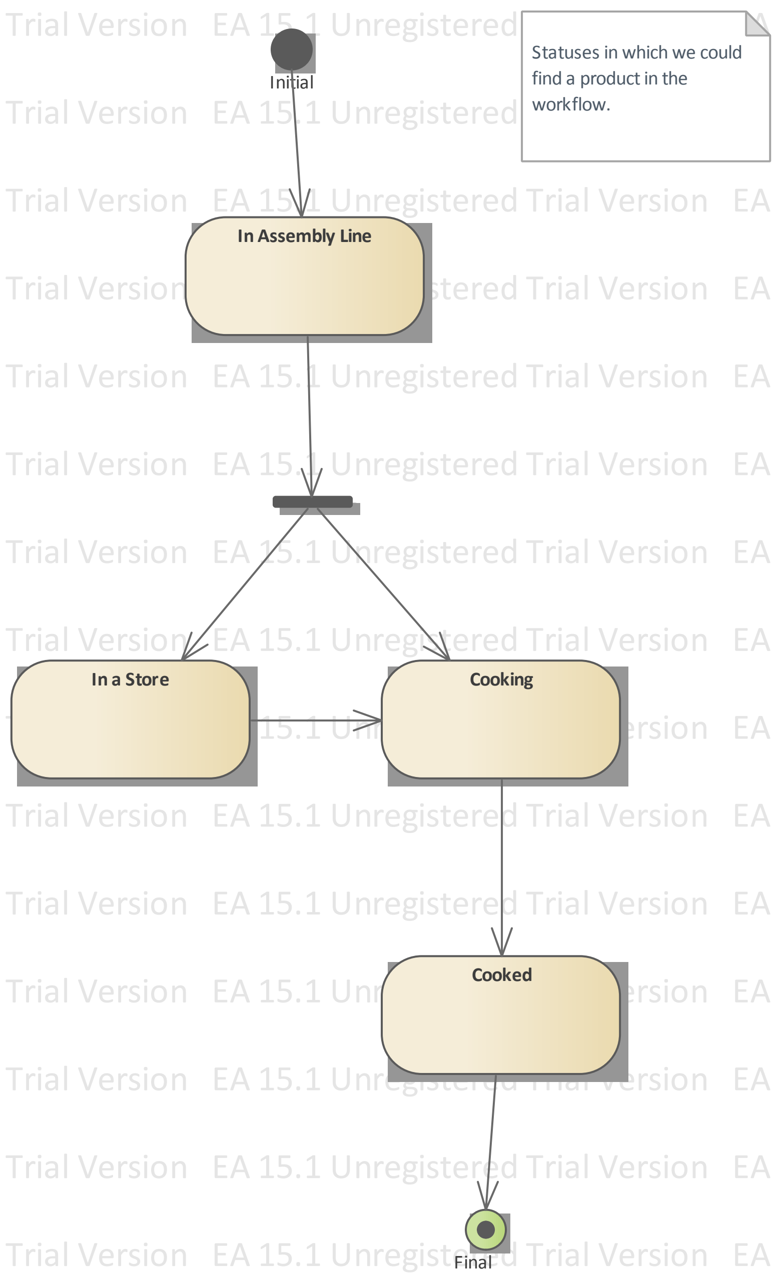
NOTE

take(ProductA) - It doesn't has a return type. If this is a "mandatory feature". Then I'll have to resort to return it the same object passed by reference.

3. State Diagrams

The following diagrams describe the possible states in which we could find a given object inside the application's workflow.

3.1 State Diagrams diagram



Trial Version EA 15.1 Unregistered Trial Version EA

8. State Diagrams

NOTE

Statuses in which we could find a product in the workflow.

4. Source Code Review

Programmer:	Ezequiel H. Martinez	**Reviewed by:**	TBD
Language:	TBD	**Date:**	06/14/2020
Version:	Alpha	**Phase:**	Development

Naming Conventions

Areas	**Comments**
Classes:	
Variables:	

Code Construction

Areas	**Comments**
Classes:	
Functions:	

Commenting Style

Areas	**Comments**
Header Comments:	
Line Comments:	

Formatting Style

Areas	**Comments**
Indentation:	
Spacing:	
Bracketing:	

General Comments:
TBD

Signed by:		/ /
	Name	Date