Work Though

Group_0434

For the project, we have eight different parts. Orders, Picking, Sequencing, Loading, Replenishing, warehouse, exception and global system.

Outline:

Orders: Orders package contains

- `ColorModelPair` class pairs the model and color.
- `FaxMachine` class receives, stores and finally transmits the order.
- `Order` class constructs orders using model and color name and stores the SKU pair.
- `Skulut` class looks up at translation table and translates the color-model pair into a SKU pair.
- `SkuPair` class stores the front and back SKU number.
- `OrderTest` class contains all the unit tests for the package Orders.

Picking: Picking package contains

- `Picker` class extends Worker class generates a picker that handles picking requests.
- `PickReqlter` class extends `RequestIter` contains a iterator for all the picking requests.
- `PickTest` class contains all the unit tests for the package Picking.

Sequencing: Sequencing package contains

- `Sequencer` class extends Worker class generates a sequencer that handles sequencing requests.
- `SequenceReqlter` class extends `RequestIter` contains a iterator for all the sequencing requests.
- `SequenceTest` class contains all the unit tests for the package Sequencing.

Loading: Loading package contains

- `Lo ader` class extends Worker class generates a loader that handles loading request that is generated by sequencer.
- `LoaderReqIter` class extends `RequestIter` class contains a iterator for all the loading request.
- `LoadTest` class contains all the unit tests for the package Loading.

Replenishing: Replenishing package contains

- `Replenisher` class generates a replenisher that picks up replenishing request.
- `ReplenishReq` class generates replenish request when stock of a certain SKU reaches below certain level.
- `ReplenishTest` class contains all the unit tests for the package Replenishing.

Exception: Exception package contains

- `IncorrectSku` class informs the user that there is a mismatch in previously and currently stored orders.
- `IncorrectWorkerStatus` class informs the user that the worker is not available at certain time.
- `InvalidTranslationTable` class informs the user that the translation table is invalid.
- `InvalidTraversalTable` class informs the user that the traversal table is invalid.
- `UndefColorModelFair` class informs the user that the color-model pair is not defined in the translation table.

WareHouse: WareHouse package contains

- `Location` class records the location, SKU, and in-stock number of certain product.
- `TraversalTable` class translates SKU to actual physical location.
- `Warehouse` class monitors the picking process.
- `WarehouseTest` class contains all the unit tests for order package.

Shared: shared package contains

- `GlobalSystem` class is the main system that monitors every behavior.
- `ListFormatter` class converts a list to a readable list.
- `Request` extends `Arraylist<String>` generates requests.
- `RequestIter` is an abstract class that contains a iterator.
- `Worker` class is an abstract class that defines all the workers

Importance:
The most important classes are all in the Shared package, it contains GlobalSystem which is the core of our project and abstract classes Worker, RequestIter. The other packages contain mostly the children of shared package.

Design patterns:
We use observer and iterator in our classes. Iterator is helping us so that different kinds of workers can iterate over requests in different order that best suit their work. In our project, we also use logger and handler to keep track of every events that happen as the system proceeding. Observer is helping us so that our program is aware of every movements.

Data
We are keeping track of a pool of workers and requests. For each one of them, we keep track of basic information (name for workers and list of items that have been requested) and state (idle/ready/busy/done for workers and pre-pick/picking/pre-sequence/sequence/pre-load/load/complete for requests).

Events
The event file is processed by global system and opened by `BufferReader`, after the file is opened, we read the events inside the event file one line at a time starting from the first line of the file. While an event is being processed, the system will determine which event it is by examining the content of the event. If an order is being recognized, the system will tell the `FaxMachine` class to accept this order and stores it in an `ArrayList`. The class `ColorModelPair` will pair the model and color of the order, class `SkuLut` will translate the model and color into SKU and the class `SkuPair` will pair up the back and front SKU of the order. `FaxMachine` will wait until there are four orders and then generate a picking request. When

the system recognizes a Picker ready event, the picker's status will change to ready and the Picker class will generate the picker. Then the `ArrayList` that contains orders will transmit to `PickReqIter` class and the system will tell picker what to pick. Every time the picker picks a fascia, the location class will keep track of the stock. If the picker picks the wrong fascia, the system will alter the picker. After the picker picked all the eight correct fascias, the `ArrayList` will transmit to `SequenceReqIter` class, and the `SequenceReqIter` class will iterate over the ArrayList and rearrange them in order. When system recognizes a Sequencer ready event, the Sequencer class will generate a new Sequencer and change his status to ready. At the same time the system will pass the `ArrayList` to the sequencer. If there is a rescan event, the sequencer will start over scanning the fascias. If the sequencer realizes there is a mismatch between the `ArrayList` and the fascias, the sequencer will set the status of this order to `PREPICK` and the next available picker will repick all the eight fascias. After the sequencer finished sequencing, the `ArrayList` will transmit to `LoadReqIter` class and a loader will be generated by loader class. If there is a rescan event, the loader will start over scanning. If there is a mismatch between the `ArrayList` and the fascias, the status of the order will become `PREPICK` and the next available picker will repick all the eight fascias. If there is a replenish event, a replenisher will be generated by replenisher class and a replenish request will be generated by `ReplenishReq` class. Then the replenisher will replenish the location indicated by the replenish request. After loader loads all the fascias in the correct order indicated by `ArrayList`, order.txt and final.csv file will be generated.

Coverage

Our code coverage is 100% under the condition that the following approved cases are ignored

- Eclemma is unable to mark functions that throw an exception as "covered".
- File reading exceptions (FileNotFoundException, IOException) are ignored.
- catch block cannot be both taken and not-taken at the same testing program.