Assignment 3

TAs: Majeed and Rafi

Publication Date: 8/12/2014 00:01
Unit Tests Deadline: 14/12/2013 23:59
Assignment Submission Deadline: 29/12/2013 23:59

-

the assignment contains many links, if you do not see them, use a different PDF reader

1 General Description

In this assignment you will simulate a real estate investment trust (REIT), BGU-REIT, which owns and operates an income-producing real estate. The company employs multiple clerks and maintenance workers. BGU-REIT receives a list of rental requests, and its task is to simulate the renting process of the requested assets. Each rental will net the company an income. Preparing a asset for a customer requires different tools and maintenance materials. Your task is to make sure that the maintenance workers have the needed tools as well as the needed maintenance materials. You will manage the simulation process starting from creating customers that deliver requests to the clerks. The clerks will try to fill the requests, by searching for available assets that meet their needs. If such a request cannot be filled, the customer is put into a waiting list, until the desired assets are freed. At the end of the rental, the customer will file a damage report and deliver it to the management, which will send the maintenance workers to fix the damages as needed. A Successful rental will net the BGU-REIT an income. At the end of the process, you will print out statistics which will show how well BGU-REIT has done. Money gained, and spent, as well as how much time each request took to complete. Please note that we explain in detail the vast majority of the system; however, we kept some sections vague to allow you more freedom in deciding how to implement them. If there is no precise explanation of a section, it means you are free to implement it as you see fit, as long as it provides the desired result.

Note: Found in: denotes that it may be found in noted class, but not limited to it. You may add the same item anywhere as you see fit.

Note: fields and methods that are mentioned below are **mandatory**, and must be included in your implementation. However, you may add **more** fields or methods, as you see fit.

2 Structures

In this section, we will detail the different **structures** found in the system, the contents of each part, and their tasks. Fields mentioned for any object are mandatory, however, it is allowed to add more, as you see fit.

2.1 Asset

Found in: Management

This object will hold information of a single asset.

An **asset** object must hold the following fields: (1) Name (2) Type (3) Location (4) Collection of AssetContent (5) Status (6) Cost per night (7) Size

1. Type: The type will be one of a list of types provided as an input file.

2. Location: Coordinates are to be stored as (x,y) coordinates in two dimensional space. The distance is to be calculated as the **Euclidean** distance between two different coordinates.

3. Status: One of three different options: (i) AVAILABLE (ii) BOOKED (iii) OCCUPIED (iv) UNAVAILABLE. Where the asset is vacant if it is not currently being used. Booked, where the asset has been booked by a customer group but not occupied yet. Occupied, if the properly is currently being used by a customer, and unavailable if the asset is not fit for renting, as long it is not repaired. Repairing a asset requires a collection of exhaustible repair materials, and different repair tools, and will be detailed later on.

4. Size: How many people the asset can fit in.

2.2 AssetContent

Found in: Asset

Asset content is an object which holds the details of one item found in the asset. An asset typically has collection of these items.

This object will hold the following fields: (1) Name (2) Health (3) Repair Cost multiplier

1. The health of the asset will deteriorate as customers use the asset. This variable will reflect the health of a single item found in the asset.

2. Repair cost multiplier: Calculating the time it will take the Maintenance Person to repair an item found in the asset equals to (100-Health)*Repair Cost Multiplier. The higher the multiplier the higher the time it takes to repair. Health begins at 100, can be fractions, and not below 0.

2.3 Assets

Found in: Management, RunnableClerk

This object will hold a collection of Asset. And contain methods that are related to assets.

There are two objects of this class:

1. Retrieve the list of damaged assets.

2. A find method which returns to the clerk an asset which fits the type and size requirements.

2.4 RepairToolInformation

Found in: Management

This object will hold information of a **single** tool type. Each tool has: (1) Name (2) Quantity.

This item comes in order to store the information of a tool that is required by an asset content item in order to get the asset content item repaired.

Example: Repairing a **Stove** might require 1 Hammer, 1 Drill, 2 Screw Drivers, then this object stores <Hammer, 1> <Drill, 1> <Screw Driver, 2> in a collection and linked to Stove.

Note: This is **not** to be stored in the Warehouse.

2.5 RepairMaterialInformation

Found in: Management

This object will hold information of a single repair material type. Each material contains the following fields: (1) Name (2) Quantity.

This item comes in order to store the information of a material that is required by an asset content item in order to get the asset content item repaired.

Example: Repairing a **Stove** might require 10 Nails, 5 Screws, 1 Sand Cloth, then this object stores <Nail, 10> <Screw, 5> <Sand Cloth, 1> in a collection and linked to Stove.

Note: This is **not** to be stored in the Warehouse.

2.6 Warehouse

Found in: Management, RunnableMaintainenceRequest

This object contains: (1) Collection of repair tools (2) Collection of repair materials.

This warehouse is the **shared** storage component, where the different maintenance persons acquire their tools and materials. You will have methods that allow the maintenance person to acquire and release tools, as well as to consume repair material.

Note**:** The warehouse **contains** enough maintenance materials and tools in order to successfully complete the simulation process. There is **no** case where your simulation will get stuck due to insufficient units.

2.7 RepairTool

Found in: Warehouse

This object is destined to be in the warehouse. It will hold the name of the tool, as well as the current quantity found in the warehouse. **E**nsure **thread safety** in this object. Why? How?

2.8 RepairMaterial

Found in: Warehouse

This object is destined to be in the warehouse. It will hold the name of the material, as well as the current quantity found in the warehouse. **E**nsure **thread safety** in this object. How is this different than Repair Tool's implementation?

2.9 Location

Found in: ClerkDetails, Asset,

A simple object which will hold the (x,y) coordinate in a 2D space and used in appropriate places in the application. Implement a method called **calculateDistance(**Location other) which accepts another location and calculates the distance between them.

Note: Since you receive as a parameter an object of the same type of the class, you **can** access its fields **directly**, even if they are private!

2.10 CustomerGroupDetails

Found in: RunnableCustomerGroupManager

This object will hold the details of the customer groups parsed from input files.

Fields: (1) Collection of Rental Requests (2) Collection of Customers (3) Group manager name

Methods: (1) addCustomer (2) addRentalRequest

2.11 ClerkDetails

Found in: RunnableClerk

This object will hold the following fields: (1) Name (2) Location.

Information parsed from the input files will be stored in this object.

This object will be wrapped by RunnableClerk and used by it.

2.12 DamageReport

Found in:

A damage report will be generated once a CustomerGroup finishes simulating its CallableSimulateStayInAsset, and the damage percentage returned, then the RunnableCustomerGroupManager will create the DamageReport object which will hold the following fields: (1) Asset (2) Damage percentage. This object will be sent to the Management for further handling.

The damage percentage in the damage report is the sum of all damage percentages returned by all the customers of the group.

2.13 RentalRequest

Found in: Management

This object will hold the information of a rental request.

Fields: (1) Id (2) Asset type (3) Asset size (4) Duration of stay (5) Asset (6) Request status

1. Asset type: The type of the asset requested by the customer group manager.
2. Asset size: The size of the asset requested by the customer group manager. Note: The asset to be booked **can** be larger than the requested size. However, be sure to choose the **smallest** one possible of the ones found.
3. Duration of stay: Duration is in days, where each day is 24 **seconds** in our simulation process..
4. Asset: The asset itself. Which was found by the RunnableClerk and will be used by the RunnableCustomerGroupManager.
5. Request status: One of four possible options:
   (a) INCOMPLETE: If the request has not been handled yet.
   (b) FULFILLED: If the request has been fulfilled, but the customer has not used it yet.
   (c) INPROGRESS: The customer is occupying the asset currently.
   (d) COMPLETE: The customer has left the asset.

Note: This object will be sent to CallableSimulateStayInAsset which will simulate the rental process of the request.

3 Passive Objects

In this section, we will detail the different **passives** found in the system, the contents of each part, and their tasks. Fields mentioned for any object are mandatory, however, it is allowed to add more, as you see fit.

3.1 Management

This object contains the following fields: (1) Collection of clerk details (2) Collection of customer group details (3) Assets (4) Warehouse (5) Collection RepairToolInformation (6) Collection of RepairMaterialInformation

Required Methods: addClerk(), addCustomerGroup(), addItemRepairTool(), addItemRepairMaterial()

Note: What is the best way to store each collection? HashMap? ArrayList? Vector? Queue? BlockingQueue? LinkedList? Ask yourself where each which.

Management simulates the system as follows:

1. Runs each Clerk as RunnableClerk in a thread. **D**o you **need** an executor? I know you **want** to! But do you **need** to?
2. Runs each CustomerGroup as RunnableCustomerGroupManager, in a thread. **D**o you **need** an executor?
3. As long as there are RentalRequests to handle:
   (a) Awaits all RunnableClerks to finish their **day**. **H**ow?
   (b) For each damage report, retrieves the asset, and executes a new RunnableMaintainenceRequest which simulates repairing the asset. **D**o you **need** an executor?
   (c) Once the handling is done, the Clerks are notified regarding the beginning of a new shift. **H**ow?

3.2 Customer

This object contains the following fields: (1) Name (2) Vandalism type (3) Minimum damage (4) Maximum damage.

Where vandalism type can be of three types: (i) ARBITRARY (ii) FIXED (iii) NONE as detailed in 2.12.

The customer calculates damage percentage to the asset. If the damage type is fixed, then the average of minimumDamage and maximumDamage is returned. The customer does not alter the health of the asset, only the RunnableCustomerGroupManager.

3.3 Statistics

Found in: Management

This object contains the following fields: (1) Money gained (2) Rentals (3) Collection of repair tools (4) Collection of repair materials.

1. Money gained: The income from rentals.
2. Rental requests: Collection of rental requests, including their information.
3. Repair tools: Tools used in the process.
4. Repair materials: Materials consumed and their quantity.

Updating this object may be done from anywhere in the program, however it is advised to do so in minimal amount of places.

4 Active Objects

4.1 RunnableClerk

This object contains the following fields: (1) Clerk details (2) Collection of RentalRequests (3) Number of rental requests.

1. Clerk details: Details of the clerk.
2. Collection of RentalRquests: Shared collection between all of the clerks found in the application, and each slot contains a RentalRequest.
3. Number of rental requests: Holds the number of the rental requests that are yet handled. **O**f what type this will be? Why is it important? Where are you going to use it?

The Clerk will simulate a day cycle in their life. Once the day begins, the clerk will take a new RentalRequest from the shared requests collection. Then, the clerk will seek and locate a asset that is suitable for the request. Once such a asset is found, the clerk will simulate going to the asset by calculating the Euclidean distance between his location and the location of the asset, and sleeping the distance in seconds.

Example: If the clerk is found at (1,1) and the asset is found at (3,1), then the Euclidean distance is 2, then the clerk will sleep for 4 (way and back) seconds. Only then, the clerk will notify the customer that the asset has been found.

As long as the total sleep time is less than 8 seconds, the clerk continues to handle requests. Once the total number of seconds passes 8, then the clerk enters wait mode until notified by the Management for a new shift. The clerk stops working when the RentalRequests to handle are down to 0.

Note: The asset found by the clerk should be stored in the RentalRequest object. This way the RunnableCustomerGroupManager can retrieve the said asset easily, and simulate the stay.

Note: The clerk changes the status of the asset to BOOKED, **prior** to going to the asset.

Note: If a suitable asset was found but is **not** AVAILABLE, then the clerk needs to **wait** for an asset to be vacated. A suitable asset will **always** be found, it just might not be AVAILABLE for you to BOOK.

4.2 RunnableCustomerGroupManager

This object contains the following fields: (1) CustomerGroupDetails

The manager retrieves the RentalRequest of the group from the CustomerGroupDetails object. As long as there are RentalRequests to handle:

1. Submits the rental request to the Management, this way it can reach the RunnableClerks.
2. Waits until the RentalRequest is fulfilled by one of the currently running RunnableClerks. **H**ow?
3. Once the request is fulfilled, it creates for each customer found in the group CallableSimulateStayInAsset.
4. Waits until all CallableSimulateStayInAsset are done. Note: You may use ExecutorCompletionService, a usage example can be seen here.
5. Combines the damage returned by each CallableSimulateStayInAsset, into one value, and **updates** the asset with the damage caused by the customers.
6. Generates a DamageReport, which is sent to the management.

This process is done until all rental requests of the group are fulfilled, and the simulation is complete.

4.3 RunnableMaintenanceRequest

Found in: Management

Note: Must be Runnable.

This object will hold the current fields: (1) Collection of RepairToolInformation (2) Collection of RepairMaterialInformation (3) Asset (4) Warehouse.

From the Asset, we retrieve the damaged contents. An asset content item is damage if its health is below 65%. Then, the total repair "cost" in **time**, is calculated, where for each item the percentage of the damage is multiplied by the repairCost multiplier found in the AssetContent. Then:

1. Acquire the required repair tools and their quantity. Note: The warehouse will **always** have enough tools.
2. Acquire the required repair materials and their quantity. Note: The warehouse will **always** have enough materials

3. **Sleep** the **cost** in **milliseconds** after rounding it to nearest long.
4. Release the acquired repair tools.
5. Mark Asset fixed. By returning the health of all its contents to 100.

You generate one RunnableMaintenanceRequest per damaged asset.

4.4 CallableSimulateStayInAsset

Found in: RunnableCustomerGroupManager

This callable will simulate the presence of **one** customer in an **asset.** After the simulation is complete, a damage percentage to the asset is calculated and returned.

- Calculating the damage percentage is done **once** per asset, per customer. Generating the percentage is done as follows, depending the vandalism type:
  1. If *ARBITRARY* then the percentage will be calculated in the range of minimumDamage and maximumDamage using randomization.
  2. If the damage type is *FIXED* then, the average of minimumDamage and maximumDamage percentage is returned.
  3. If the damage type is *NONE* then, the regular tear and wear percentage will be returned: *0.5%*
- Sleeping is done by converting each 24 hours, to 24 seconds.

5 Program Cycle

First, you need to parse 4 files and create objects to store the information parsed from these files. These files include information regarding the different parts of the system, full information can be seen at 8

Once the parsing process is done. You will launch the simulation process, and call the Management.

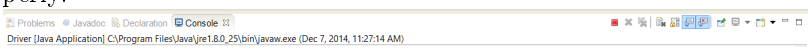
The Management will launch the CustomerGroupManagers, Clerks and Maintenance Persons. Where each CustomerGroup contains a collection of its rental requests. The clerks will take rental requests from the queue, fulfill such a request, and provide it to the Customer. The customer will stay in the properly, and in the end of their stay, generate a damage report, which will be sent to the Management. The management provides the damage report to one of its Maintenance people for repair. Maintenance people will repair the assets, and the cycle continues.

The application will wait for the process to end. Then and only then, your application must shut down completely. Be sure to shutdown all your executors, as well as threads that may be working in your application.

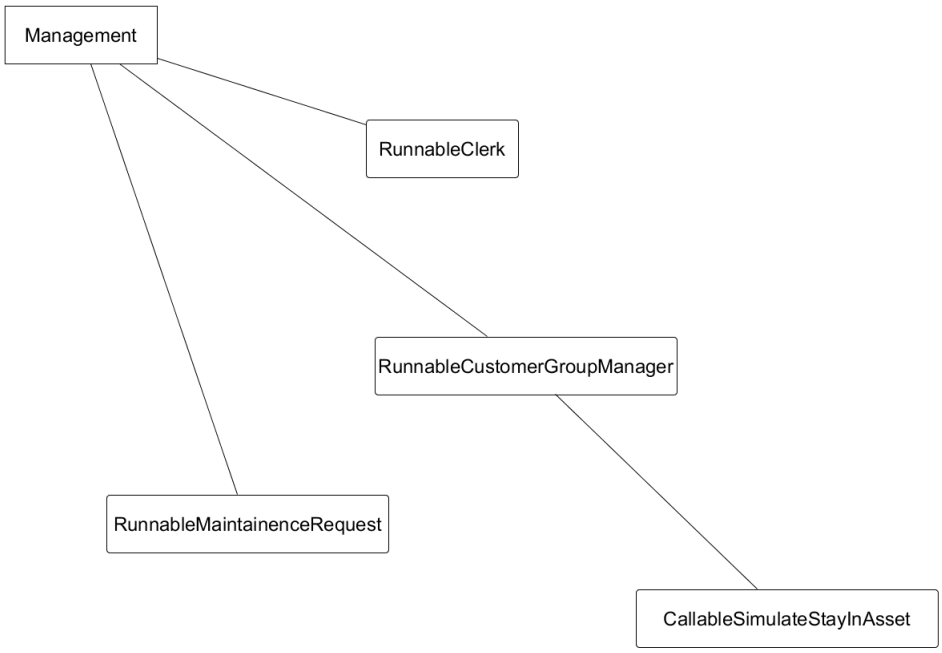
6 Shutdown Process

Remember: Exiting your main does not make any other threads end.

You need to allow for graceful shutdown once the application completes its life cycle. When the manager knows that the last rental request has been completed you must print out the contents of the Statistics object, and shutdown your application. Which means you need to handle shutting down all the threads found in your application, as well as shutting down all the executors you may be using, and it must be done gracefully. Meaning no abrupt exits of threads. The thread flow must reach the end. Whe you are done, do you see the red rectangle below? Then your application did **not** shutdown properly!



7 Relations Diagram



8 Input Files and Parsing

8.1 Input Files

You *may* assume correct input. You *may* assume the order of information is as you see in this example. There are 4 different input files.

1. InitialData:
   (a) Warehouse which consists of a list of tools and a list of materials.
   (b) Staff which consists of clerks and the number of the maintainence persons.

```
<REIT>
        <Warehouse>
                <Tools>
                        <Tool>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Tool>
                            ⋮
                        <Tool>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Tool>
                </Tools>
                <Materials>
                        <Material>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Material>
                            ⋮
                        <Material>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Material>
                </Materials>
        </Warehouse>
        <Staff>
                <Clerks>
                        <Clerk>
                                <Name></Name>
                                <Location x="" y=""/>
                        </Clerk>
                            ⋮
                        <Clerk>
                                <Name></Name>
                                <Location x="" y=""/>
                        </Clerk>
                </Clerks>
                <NumberOfMaintenancePersons></NumberOfMaintenancePersons>
                <TotalNumberOfRentalRequests></TotalNumberOfRentalRequests>
        </Staff>
</REIT>
```

2. AssetContentsRepairDetails: This file contains information regarding the different items may be found in assets, and the required tools and materials in order to repair the asset due to customer vandalism.

```
<AssetContentsRepairDetails>
        <AssetContent>
                <Name></Name>
                <Tools>
                        <Tool>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Tool>
                           ⋮
                        <Tool>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Tool>
                </Tools>
                <Materials>
                        <Material>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Material>
                           ⋮
                        <Material>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Material>
                </Materials>
        </AssetContent>
                ⋮
        <AssetContent>
                <Name></Name>
                <Tools>
                        <Tool>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Tool>
                           ⋮
                        <Tool>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Tool>
                </Tools>
                <Materials>
                        <Material>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Material>
                           ⋮
                        <Material>
                                <Name></Name>
                                <Quantity></Quantity>
                        </Material>
                </Materials>
        </AssetContent>
</AssetContents>
```

3. Assets: This file contains a list of Assets, and their information.

   Format:

```
<Assets>
        <Asset>
                <Type></Type>
                <Size></Size>
                <Location x=" " y=" "/>
                <CostPerNight></CostPerNight>
                <AssetContents>
                        <AssetContent>
                                <Name></Name>
                                <RepairMultiplier></RepairMultiplier>
                        </AssetContent>
                                ⋮
                        <AssetContent>
                                <Name></Name>
                                <RepairMultiplier></RepairMultiplier>
                        </AssetContent>
                </AssetContents>
        </Asset>
            ⋮
        <Asset>
                <Type></Type>
                <Size></Size>
                <Location x=" " y= " "/>
                <CostPerNight></CostPerNight>
                <AssetContents>
                        <AssetContent>
                                <Name></Name>
                                <RepairMultiplier></RepairMultiplier>
                        </AssetContent>
                                ⋮
                        <AssetContent>
                                <Name></Name>
                                <RepairMultiplier></RepairMultiplier>
                        </AssetContent>
                </AssetContents>
        </Asset>
</Assets>
```

4. CustomerGroups: This file contains a list of customer groups, their rental requests, the customers, and their information.

   Format:

```
<Customers>
        <CustomerGroups>
                <CustomerGroupDetails>
                        <GroupManagerName></GroupManagerName>
                        <Customers>
                                <Customer>
                                        <Name></Name>
                                        <Vandalism></Vandalism>
                                        <MinimumDamage></MinimumDamage>
                                        <MaximumDamage></MaximumDamage>
                                </Customer>
                                        ⋮
                                <Customer>
                                        <Name></Name>
                                        <Vandalism></Vandalism>
                                        <MinimumDamage></MinimumDamage>
                                        <MaximumDamage></MaximumDamage>
                                </Customer>
                        </Customers>
                        <RentalRequests>
                                <Request id=" ">
                                        <Type></Type>
                                        <Size></Size>
                                        <Duration></Duration>
                                </Request>
                                    ⋮
                                <Request id=" ">
                                        <Type></Type>
                                        <Size></Size>
                                        <Duration></Duration>
                                </Request>
                        </RentalRequests>
                </CustomerGroupDetails>
                        ⋮
```

```
                    ⋮
            <CustomerGroupDetails>
                    <GroupManagerName></GroupManagerName>
                    <Customers>
                            <Customer>
                                    <Name></Name>
                                    <Vandalism></Vandalism>
                                    <MinimumDamage></MinimumDamage>
                                    <MaximumDamage></MaximumDamage>
                            </Customer>
                                    ⋮
                            <Customer>
                                    <Name></Name>
                                    <Vandalism></Vandalism>
                                    <MinimumDamage></MinimumDamage>
                                    <MaximumDamage></MaximumDamage>
                            </Customer>
                    </Customers>
                    <RentalRequests>
                            <Request id=" ">
                                    <Type></Type>
                                    <Size></Size>
                                    <Duration></Duration>
                            </Request>
                                ⋮
                            <Request id=" ">
                                    <Type></Type>
                                    <Size></Size>
                                    <Duration></Duration>
                            </Request>
                    </RentalRequests>
            </CustomerGroupDetails>
    </CustomerGroups>
</Customers>
```

8.2 Parsing

Parsing is the *act* of reading text and converting it into a more useful in-memory format, "understanding" what it means to some extent. XML parsers for example, parse XML-formatted files. HTML parsers, parse HTML-formatted files. In our case, we wish to parse XML files of the format detailed above.

The given files follow a *strict* XML structure, and the correctness of input is assumed, which adds to the simplicity of the parsing process. Be sure to read Java XML Parser API and manual before attempting to parse any files.

You must implement a class called "Driver" which implements your "main" function. It receives the file names as arguments, calls the appropriate static functions to parse the different files and create the needed objects, and once all the needed objects are created, the main function executes the simulation process (Management). Your Management is **not** to parse any files. Your parsing **must** be done in *one* class only. Parsing may be done using multiple static functions in a new **static** class which contains the different static parsing functions, and it must be called from your main before launching

the simulation process. Your main is **not** to parse any files. It just receives the objects and launches the simulation process. **W**hy?

9 toString()

All objects found in the Statistics object must implement a toString() method which returns a concatenated string of the object's fields. When running the toString() method of the Statistics object, a printout of all its contents in a human readable manner needs to be returned.

StringBuilder is available for your disposal in order to concatenate elements in a loop. Using "+" is **bad**. **W**hy?

When you wish to print the contents of the Statistics object you hold, all you need to do is to print the result of the toString() method of the Statistics object, and inside it, you will run the toString() method of all the objects contained in it, concatenate, and return.

10 Mandatory Requirements

You are *required* to use *all* the following in your application:

1. Blocking Queue
2. Semaphore
3. Guarded Blocks: What happens when you notify(), and the thread you wish to notify() is **not** in wait mode? Is this a problem? What about notifty() vs notifyAll(), when to use each?
4. CountDownLatch
5. Executor Service: Executors come to **manage** threads. At every point where you wish to use an executor, ask yourself. Do I **need** an executor to manage my threads? If the answer is **no.** Do **not** use an executor!
6. Threads
7. Atomic: Java provides atomic objects which come in order to assist you keeping thread safety. Use at least one of these.

   You need to be ready to explain where have you used each one of them, as well as explain the reasons behind your decision.

Also, note the differences between: ArrayList and Vector. Be sure to understand which one of the two fits better your needs in each situation.

11 Application Progress: Logger

Logging is the process of writing log messages during the execution of a program to a central place. This logging allows you to report and persists error and warning messages as well as info messages (e.g. runtime statistics) so that the messages can later be retrieved and analyzed. Loggers are a great asset especially for a thread oriented application. See guide here, on how to use a logger appropriately.

Your application needs to implement a logger mostly for informative reasons, and to follow the progress of the application and its different threads. Please store logging data to a file in order to explain the progress of your application to the grader. Make your logger print informative and readable text, so it is easier to follow and understand. The better your logger is, the easier the grader can understand your application. So be sure to invest some time to decide where and what to print your logging information.

12 Compiling the Application: Another Neat Tool or ANT

Did you love makefile? Great! We're going to do the same thing for Java.

You are required to create an Ant file for your application in order to complete it. Ant file is equivalent to makefile that you have learned in class. You are required to read these two small guides:

- What is ant?
- Ant Tutorial

For further, and complete information regarding ANT, you may refer to Ant Manual.

Once you are done reading, and understanding, what Ant is all about, you are required to create an ant build file for your application.

The only mandatory target for Ant build file is "run": *ant run -Darg0=InitialData.xml -Darg1=CustomerGroups.xml -Darg2=RentalRequests.xml -Darg3=Assets.xml -Darg4=AssetContents.xml"*. When running ant, you need to compile your application and run it using the showed above input files. The order of the files is ***important***.

This is a **mandatory** requirement, your application must compile without warnings when launched via "ant".

Compiling via Eclipse alone is **unacceptable**!

13 Deadlocks, Waiting, Liveness and Completion

Deadlocks You should identify possible deadlock scenarios and prevent them from happening. Be prepared to explain to the grader what possible deadlocks you have identified, and how you have handled them.

Waiting You should understand where wait cases may happen in the program, and how you've solved these issues.

Liveness Locking and synchronization are required for your program to work properly. Make sure not to lock too much so that you don't damage the liveness of the program. Remember, the faster your program completes its tasks, the better.

Completion As in every multi-threaded design, special attention must be given to the shut down of the system. When all orders are sent to be executed and nothing is waiting in queue, we will need to shut down the program. This needs to be done gracefully, not abruptly.

14 Unit Tests

You are required to write tests and a design by contract interface for the *Warehouse* class, as well as unit testing, according to the test driven development rules. Please note that your submission does not have to include a full implementation of the *Warehouse* class, but only an empty one. Submission of this part is due 14/12/2014, 23:59.

The following files are needed:

1. Warehouse interface.
2. Warehouse empty implementation of the interface.
3. Unit Test for the warehouse class.

If you wish to add new helping methods in order to successfully do your tests. You must **not** change the interface, and you must **not** change the implementation. What you need to do is to extend the Warehouse class (e.g. WarehouseTesting extends Warehouse) and in the WarehouseTesting class you may add the new methods. Then you create a Unit Test class which tests WarehouseTesting and not Warehouse. Since WarehouseTesting includes your new added methods, everything can be tested correctly. Since WarehouseTesting contains what Warehouse has, you in the end, by testing WarehouseTesting class, you check Warehouse class as well.

You are not required to test synchronization in your class.

15 Documentation and Coding Style

There are many programming practices that **must** be followed when writing any piece of code.

- Follow Java Programming Style Guidelines. A part of your grade will be checking if you have followed these mandatory guidelines or not. It is important to understand that programming is a team oriented job. If your code is hard to read, then you are failing to be a productive part of any team, in academics research groups, or at work. For this, you must follow these guidelines which make your code easier to read and understand by your fellow programmers as well as your graders.
- Do *not* be afraid to use long variable and method names as long as they increase clarity.
- Avoid code repetition - In case where you see yourself writing same code block in two different places, it means this code must be put in a private function, so both these places may use it.
- Full documentation of the different classes and their public and protected methods, by following Javadoc style of documentation. You may, if you wish, also document your local methods but they are not mandatory.
- Add comments for code blocks where understanding them without, may be difficult.
- Your files must *not* include commented out code. This is garbage. So once you finish coding, make sure to clean your code.
- Long functions are frowned upon (30+ lines). Long functions mean that your function is doing *too many tasks*, which means you can move these tasks to their own place in private functions, and use them as appropriate. This is an important step toward increased readability of your code. This is to be done *even* in cases where the code is used **once**.
- Magic numbers are **bad**! Why? Your application must not have numbers throughout the code that need deciphering.
- Getters and Setters are **bad**! They **break** encapsulation. If your object is **not** a data structure object, then you must avoid using setters and getters at all costs! Read why here, here, here, here, and here
- Do **not** send collection of items to constructors. They **can** reveal internal implementation. Instead, create a method for the object which **adds** one item to the object. Example: create addClerk, or addCustomerGroup methods in the Manager class, so you can add new clerks or customer groups on demand.
- Do *not* reveal the internal implementation of your objects.

- For instance: You wish to add an RentalRequesst object to a collection object inside Management object. You do not expose (to other classes) the inner collection object (by using a getter) then add the RentalRequesst to it, but instead, you send the RentalRequesst to the Management object, which implements a method (such as: addRentalRequesst(RentalRequesst rentalRequest)) that adds the RentalRequesst to the collection. This is to be done for all access attempts to internal data of objects. You do not retrieve the internal data from that object, instead, you implement a method for that object which fulfills your desire.
- Another example: You wish to calculate the distance between two locations? You do not retrieve the (x, y) values from each Location object and do the calculation yourself, but instead implement a function called calculateDistance in Location class, which takes another Location object and calculates the distance internally, and returns the desired value. Note: Since you are accessing the same class, you **can** access the private fields **directly**!

Your implementation must follow these steps in order to keep the code ordered, clean, and easy to read.

16 Submission Instructions

- Submission is done *only* in pairs. If you do not have a pair, find one. You need explicit authorization from the course staff to submit without a pair. You cannot submit in a group larger than two.
- You *must* submit one .tar.gz file with all your code. The file should be named "assignment3.tar.gz". Note: We require you to use a .tar.gz file. Files such as .rar, .zip, .bz, or anything else which is not a .tar.gz file will *not* be accepted and your grade will suffer.
- Your tar.gz file *must* include an ant build file configured for your application. The grader will extract your file, build your application using ant build fie, and execute it.
- Extension requests are to be sent to majeek. Your request email must include the following information:
  - Your name and your partner's name.
  - Your id and your partner's id.
  - Explanation regarding the reason of the extension request.
  - Official certification for your illness or army drafting.

  Requests without a *compelling* reason will not be accepted.

17 Grading

Although you are free to work wherever you please, assignments will be checked and graded on Computer Science Department Lab Computers - so be sure to test your program thoroughly on them before your final submission. "But it worked fine on my windows based home computer" will not earn you any mercy points from the grading staff, if your code fails to execute at the lab.

Grading will tackle many points, including but not limited to:

- Your TDD design and implementation.
- Your application design and implementation.

- For each point in 10, where have you implemented it, and the reasons behind your decision.
- Automatic tests will be run on your application. Your application must complete successfully and in reasonable time.
- Liveness and Deadlock, causes of deadlock, and where in your application deadlock might have occurred, without your solution to the problem.
- Synchronization, what is it, where have you used it, and a compelling reason behind your decisions.
- Checking if your implementation follows the guidelines detailed in 15.

18 Questions and Assistance

- forum for the assignment.
- We will *not* answer emails related to the assignment. Please refrain from sending them.
- F.A.Q will be updated at the website of assignment 3. It is **mandatory** to read the F.A.Q ***daily*** for updates. All changes noted there are binding.
- We will visit the labs and help the students at the times published on the website.
- ENJOY! :-)