Concurrent and Distributed Systems

Grand Sorcerer - Save the world

Final-Homework

January 12, 2020

Student: Marcu Andrei Cristian

Computers and Information Technology

CEN 3.2 A

3rd Year

Application design

The project is written in java. The common code for all the task is formed of 12 files ***Coven.java CovenHelper.java CovenReset.java Demon.java DemonSpawner.java GrandSorcerer.java GrandSorcererHelper.java Main.java PotionTransfer.java Undead.java Witch.java WitchHelper.java***.
We will talk about each of them later on the presentation. Now let's stick on the overall application design
I've respected the classes presented in the problem statement(Coven, Demon, DemonSpawner, GrandSorcerer, GrandSorcererHelper, Undead, Witch) . I've created four extra classes(CovenHelper, CovenReset, PotionTransfer and WitchHelper). CovenHelper and WitchHelper keep the predefined data of the coven and witch, in order to make the main class more readable, avoiding useless code and keep only it's main functionality. CovenReset is a class that sleeps for 10 seconds and will reset the ingredients in coven after the sleep time finishes. The PotionTransfer is a class that simulates a "transfer route" between Witch and GrandSorcerer. We can send/receive potions through this class.
Now I am going to present each class and the methods present in them

The Coven class

Implements a thread that acts like a coven.
It contains the following variables:

- *ingredientsInCoven* - it's a ConcurrentHashMap ¡ String, Integer ¿ which contains the ingredients present in coven. The key is the ingredient name and the value is the number of pieces of that ingredient.
- *covenHelper* - it's an instance of the CovenHelper class which creates the empty map presented above, the random value of N, and defines the ingredients name
- *coven* - it's the coven matrix. contains 0 if the spot is unoccupied or demonID if there is a demon
- *covenID* - unique identifier of the coven
- *demons* - ArrayList which contains all the demons present in the coven
- *witchesInCoven* - ArrayList which contains all the witches present in coven

- *undeadsInCoven* - ArrayList which contains all the undead present in coven
- *covenReset* - it's an instance of the CovenReset class which will reset the ingredientsInCoven map every 10 seconds as said in the problem statement
- *grandSorcerer* - it's the only instance of the Grand Sorcerer. We use it to access a boolean variable from the GrandSorcerer class(the gameEnded variable) which tells us when to stop the Coven thread execution

It contains the following synchronization mechanisms:

- *ingredientsLock* - it's a ReentrantLock used everytime we modify the ingredientsInCoven map so no other thread can access it while it's being modified
- *demonsLock* - it's a ReentrantLock used everytime we modify the demons arraylist so no other thread can access it while it's being modified
- *covenLock* - it's a ReentrantLock used everytime we modify the coven matrix so no other thread can access it while it's being modified
- *undeadsInCovenLock* - it's a ReentrantLock used everytime we modify the undeadsInCoven arraylist so no other thread can access it while it's being modified

It contains the following methods:

- *Coven constructor* - it's the class Contructor which creates the instance of CovenHelper, initializes the grandSorcerer with the only instance of it, initializes the demons ArrayList, witchesInCoven ArrayList, undeadsInCoven ArrayList, the coven matrix, the ingredientsInCoven map, sets the covenID of the coven and creates the covenReset instance of the CovenReset class
- *public void run()* - it's the thread method which starts the covenReset thread. This function will run until the game ends. We also print the ingredients present in coven every 5 seconds.
- *public boolean addDemon(Demon demon)* - it's the function that tried to add a demon in the coven. It returns false if the demon add failed and true if the demon was added successfully. The parameter is the demon we are trying to add. The demon's X and Y are random values. If the random spot is free in the coven matrix we place the demon's ID in the matrix, we add the demon in the demons ArrayList and we start the execution of the demon thread. This all happens with the demonsLock locked because we don't want demons to take the position while we add the demon. So while we add a new demon the demons can't move.

- *public void moveDemon(Demon demon)* - it's the function that moves a demon in random direction. The parameter is the demon we are going to move. We pick a random number from [0,4). 0 means right 1 means up, 2 means down, 3 means left. If the random movement direction picked is not possible we pick another. We move the demon to the new position, we call the createIngredient() function and we modify the X and Y of the demon inside the class using the moveDemon method. We also treat the case when the demon is surrounded(he can't move in any direction). All those actions happen with the covenLock locked because we don't want the demons to move in the matrix while we search for a free spot. We might find a free spot and another demon take it while we modify it.

- *public void createIngredient(Demon demon)* - it's the function that creates ingredients. When a demon moves he creates ingredients according to it's social skill (if there are no penalty rounds). If there is a penalty round we don't create any ingredients and decrease the number of penalty rounds. The maximum number of ingredients the demon can create is 10. The ingredients the demon will create are picked random. We modify the value in the ingredientsInCoven map with the previous value + 1. All these actions are done with the ingredientsLock locked, because we access the values in the map and save them while we modify the map, we don't want those values to be modified after we saved them.

- *public boolean canMoveLeft(Demon demon)* - it's the function that checks if the demon can move in the left direction but this function also treats more cases. It returns false if the demon can't move and true if the demon can move. The demon we want to move is passed as parameter. In this function we also check if the demon hits a wall(and increase the penalty rounds, we make sure he doesn't hit the wall 2 times in a row, and we count the total walls hit because with each 10 walls hit we decrease it's social skill with 50). We also check if he meets another demon and increase both of them social skill with 50

- *public boolean canMoveDown(Demon demon)* - same as above but for down direction

- *public boolean canMoveUp(Demon demon)* - same as above but for up direction

- *public boolean canMoveRight(Demon demon)* - same as above but for right direction

- *public void lose10PercentIngredients()* - it's the function that reduces the number of ingredients by 10 percent. This happens when a witch can't defeat an undead. This effect happens under the ingredientsLock because we will modify the ingredientsInCoven map.

The CovenHelper class

Implements a support class for the Coven class presented above. It contains the following variables:

- *ingredients* - it's an array of strings which contains predefined ingredients names
- *ingredientsInCoven* - it's the map used in coven. we will initialize it in the constructor of this class
- *coven* - it's the coven matrix. we will initialize it in the constructor of this class
- *random* - instance of Random(used for N value)
- *i, j* - used to iterate through the matrix

It contains the following methods:

- *CovenHelper constructor* - initializes the ingredientsInCoven map with 0 for each ingredient, generates a random N which is the length of the square matrix and initializes the matrix with 0 on each position

The CovenReset class

Implements a thread-support class for the Coven class presented above. It contains the following variables:

- *coven* - it's the coven this Thread will reset

It contains the following methods:

- *CovenReset constructor* - we initialize the coven
- *public void run()* - it's the thread specific method which runs until the game ends(all undeads are defeated). This function will reset the coven's ingredientsInCoven map every 10 seconds. It uses the ingredientsLock lock defined in the coven because we modify the value of the ingredientsInCoven map. It also scares all the demons in the coven making them stop for 1 second.

The Demon class

Implements a thread that acts like a demon.
It contains the following variables:

- *demonID* - unique identifier of the demon
- *X* - line position in matrix
- *Y* - column position in matrix
- *coven* - the coven that demon belongs to
- *penaltyRounds* - the number of penalty rounds(he can't create ingredients)
- *socialSkill* - the social skill of the demon. with each 100 social score the demon can create one more ingredient with a maximum of 10
- *wallsHit* - counter increased each time the demon hits a wall. With 10 walls hit this value is reset to 0 and the social skill is decreased with 50
- *isRetired* - it's a boolean that causes the thread to stop when the demon is scared by the undead
- *isScared* - it's a boolean that causes the demon to sleep for 1 second after the coven ingredients are reset(every 10 seconds)
- *canUp, canDown, canLeft, canRight* - are booleans used to avoid hitting a wall 2 times in a row

It contains the following methods:

- *Demon constructor* - initializes the demonID, the coven, the X and Y
- *public void run()* - the thread specific method. works until the demon is retired(by an undead). this function moves the demon, decreases it's social skill by 50 when he hits 10 walls every 30 ms. Also checks when the demon is scared and puts it to sleep for 1 second
- *public void currentPosition()* - prints the demon's current position in matrix
- *public void moveDemon(int newX, int newY)* - modifies demon's X and Y. The parameters are the new values. This function is called each time the demon moves.

- *public void demonSurrounded()* - sends the demon to sleep for a random time between 10 and 50 ms. this is called when the demon has no direction to move
- *public void retireDemon()* - this method retires a demon removing it from the demons ArrayList and coven matrix. We also mark the boolean isRetired as true and stop the thread execution. All those actions happen under the lock of the covenLock and demonsLock. We are modifying the coven matrix(that's why we use covenLock) and the demons ArrayList(that's why we use demonsLock).

The DemonSpawner class

Implements a thread class which spawns demons every 500-1000 ms
It contains the following variables:

- *coven* - the coven where the demon spawner is placed
- *demonsSpawned* - it's a static variable which is increased everytime a demon is spawned. We use this to give each demon an unique id.

It contains the following synchronization mechanisms:

- *demonsSpawnedLock* - it's a ReentrantLock used everytime we increase the demonsSpawned value

It contains the following methods:

- *DemonSpawner constructor* - initializes the coven
- *public void run()* - thread specific function. Works until the undeads are defeated. Spawns a demon every 500-1000 ms by calling the spawnADemon function
- *private void spawnADemon()* - picks a random position for X and Y. Creates a new instance of the Demon class and adds it with the coven's addDemon function and increases the demonsSpawned number. While we add the demon in the coven matrix we lock it with the covenLock. While we increase the demonsSpawned number we lock with the demonsSpawnedLock

The GrandSorcerer class

Implements a thread class which represents the grand sorcerer
It contains the following variables:

- *potions* - is an arraylist that contains all the potions the grand sorcerer has
- *covens* - vector of covens containing all covens
- *witches* - vector of witches containing all witches
- *undead* - vector of undeads containing all undeads
- *covensNo* - number of covens(random from 3 to 20)
- *witchesNo* - number of witches(random from 1 to 10)
- *rand* - Random instance used for generating random numbers
- *witchHelper* - the only WitchHelper class instance
- *potionTransfer* - the only PotionTransfer class instance
- *grandSorcererHelper* - the only GrandSorcererHelper class instance
- *gameEnded* - boolean that will be true when all undeads are defeated. the initial value is false

It contains the following synchronization mechanisms:

- *potionsLock* - it's a ReentrantLock used everytime we add / take potions from the sorcerer

It contains the following methods:

- *GrandSorcerer constructor* - generates random number for covens(from 3 to 20) , of witches (from 1 to 10) and undeads (from 20 to 50) . Creates the potionTransfer, witchHelper, potions and potionsLock. Created and start the covens, witches and undeads
- *public void run()* - thread specific function. Prints the number of potions the Grand Sorcerer has every 2 seconds. Also checks if all undeads are defeated. If all undeads are defeated it stops the program by making the gameEnded boolean true. When we print the number of potions we lock the potionsLock so no potions are going to be added/taken while we print.

The GrandSorcererHelper class

Implements a support class GrandSorcerer class presented above
It contains the following variables:

- *grandSorcerer* - the unique instance of GrandSorcerer used to access the covens and their number
- *demonSpawners* - is an array of DemonSpawner . Contains all the spawners

It contains the following methods:

- *GrandSorcererHelper constructor* - creates and starts the demon spawners one in each coven

The Main class

Creates and start the GrandSorcerer instance. This starts the cicle of execution

The PotionTransfer class

Implements a thread class which represents the grand sorcerer
It contains the following variables:

- *grandSorcerer* - the unique GrandSorcerer instance
- *rand* - Random instance

It contains the following methods:

- *public void givePotion(String potion)* - the parameter is the potion we are giving. This function sends the potion from the witch to the Grand Sorcerer is he has less than 20 potions. While we are doing this the potionsLock is locked so the number of potions the Grand Witches has won't be modified by another witch when she adds a potion
- *public int takePotion()* - return 0 if the Grand Sorcerer doesn't have enough potions for the witch or returns the number of potions the witch took if the Grand Sorcerer had enough. We pick a random number of

potion from 2 to 5. We add the potions in an auxiliary arraylist and after we pick enough potions we iterate through the auxiliary array and delete all the potions from the main arraylist. The potionsLock is locked here too while we modify the potions ArrayList.

The Undead class

Implements a thread class which represents undeads
It contains the following variables:

- *undeadNo* - static int that has the total number of undeads
- *covens* - array of covens. we need this to pick a random one to haunt
- *covensNo* - the number of covens
- *undeadID* - undead unique identifier
- *coven* - the coven we will haunt next
- *retiredDemons* - an integer picked randomly from 5 to 10. It's the number of demons the undead will retire if the coven is not defended by a witch
- *isDefeated* - boolean variable which is initially false. If a witch defeates an undead this turns to true and stops the thread execution
- *rand* - Random instance

It contains the following methods:

- *Undead constructor* - intializes the undeadID, rand, covens and covensNo
- *public void run()* - works until the undead is defeated by a witch. The undead picks a random coven to haunt. if there is no witch in coven the undead will scare a random number of demons(from 5 to 10) and reset all ingredints in coven . After 500-1000 ms the undead removes himself from the actual coven and goes to haunt another random coven
- *public int pickACoven()* - method that picks a random coven and returns it
- *public void scareDemons(int numberOfRetiredDemons* - the parameter is the number of demons we are going to retire. We add demons in an auxiliary array. When we have enough demons we iterate through the auxiliary array and remove them from the main demons array. We use the demonsLock when we access the demons ArrayList

- *public void loseAllIngredients* - it's a method that makes all ingredients in coven with value 0. Uses the ingredientsLock because we modify the ingredientsInCoven map. We use the default map from the covenHelper for the reset.

The Witch class

Implements a thread class which represents witches
It contains the following variables:

- *witchID* - witch unique identifier
- *covens* - array with all the covens so we can pick one to defend
- *covensNo* - the number of covens
- *coven* - the coven the witch is going to defend next
- *potionsTook* - the number of potion the witch will take in the fight
- *rand* - Random instance
- *witchHelper* - the unique WitchHelper instance
- *potionTransfer* - the unique PotionTransfer instance

It contains the following synchronization mechanisms:

- *witchLock* - lock used when the witch fights an undead. we don't want 2 witches fight the same undead

It contains the following methods:

- *Witch Constructor* - initializes the witchLock, witchID, covens, covensNo, rand, witchHelper and potionTransfer
- *public void run()* - runs until all the undeads are defeated. the witch picks a random coven to defend. If in the coven the witch defend is an undead they're going to fight. If the witch can take enough potions from the grand sorcerer she kills the undead stopping the undead thread and decreasing the total number of undeads. If the witch can't take enough potions from the grand sorcerer the coven loses 10 percent of all ingredients. When the witch fights the witchesLock is locked. Witch stays in a coven for 10-30 ms after the sleep time she leaves the current coven and haunts another one

- *public int pickACoven()* - returns an integer which is the randomly picked coven
- *public void createPotion(int covenNo)* - the parameter covenNo is the coven we are taking ingredients from. We create potion only if the grand sorcerer has less than 20 potions. We are using an integer priority which is initially 8. First we try to make any potion with 8 ingredients. If we don't have any ingredients for a potion with 8 we decrease the propriety to 7 and so on until priority is 3 or a potion was created. If we create a potion we decrease the ingredients used by -1. We send the potion to the sorcerer. The ingredientsLock from the coven is locked while we decrease the ingredientsInCoven map values.
- *public void sendPotion(String potion)* - the parameter potion is the potion we are going to send. We are calling the potionTransfer givePotion to send the potion to the grand sorcerer
- *public int takePotions()* - returns the number of potions the witch takes. if the return value is 0 the Grand Sorcerer couldn't give the witch enough potions. It calls the potionTransfer method takePotion.

The WitchHelper class

Implements a support class for the Witch class
It contains the following variables:

- *potions* - map containing the potion recipes. Key = potion name and value = Array of ingredients
- *ingredientsForPotion* - an array used to insert ingredients in the potions map
- *potionNames* - predefined potion names
- *covenHelper* - an instance of CovenHelper used to access the ingredients

It contains the following methods:

- *WitchHelper Constructor* - calls the createPotionNames function and creates the potions map.
- *public void generateRandomIngredients()* - picks a random number of ingredients from 4 to 8. We have a float random number called change which gives each ingredient a 50 percent chance to be picked in the recipe. We save the ingredients picked in an auxiliary array called randomIngredientsForPotion. When we have enough ingredients picked we save the

auxiliary array in the ingredientsForPotion array which will be added in map by the constructor

- *public void createPotionNames()* - fills the potionNames array with predefined potion names

Synchronization methods used

- *Common code*
  - ingredientsLock used for the ingredientsInCoven map from the Coven class
  - demonsLock used for the demons array from the Coven class
  - covenLock used for the coven matrix
  - undeadsInCovenLock used for the undeadsInCoven array
  - demonsSpawnedLock used for the demonsSpawned integer from the DemonSpawner class
  - potionsLock used everytime we send/receive potions. It's defined in the GrandSorcerer class
  - witchLock used when a witch is fighting an undead so no 2 witches are going to fight the same undead. Defined in the Witch class
- *Task 1*
  - demonRetireSemaphore defined in the GrandSorcererHelper class. This semaphore releases a permit every 50 ms and a random demon will pick it, making him retired.
- *Task 2*
  - sleepingDemonsNumberSemaphore defined in the Demon class. We acquire it when the demon reaches the main diagonal and we release it after the demon is sleeping. This semaphore is also used when we wake up all demons.
- *Task 3*
  - demonsBarrier is a CyclicBarrier with N/2 parties. We have one for each coven. The demons await when they reach the main diagonal. When N/2 demons await the barrier is lowered and they woke up.

- *Task 4*
  - demonsBarrier is a self-made CyclicBarrier. Works exactly like the one presented before. This barrier is defined in the CyclicBarrier class.

Extra tasks

- *Task 1*
  - I've created a new class called DemonRetire. This class uses the semaphore defined in the GrandSorcererHelper class. Releases a permit every 50 ms. The demon run method tries to acquire this semaphore. Every 50 ms an unknown demon will acquire the semafore and begin retired.
- *Task 2*
  - I've added a new if in the demon class which checks if the demon is sleeping. I've added a semaphore which keeps the order of modifying the map sleepingDemonsNumer. The map sleepingDemonsNumber has the key = coven number and value the number of demons sleeping in the coven number. The demons start sleeping when they reach the main diagonal. When all demons are on the main diagonal we get on the else branch of the if, make the all demons isNotSleeping boolean again true, and modify the value in the map to 0.
- *Task 3*
  - I've added a CyclicBarrier called demonsBarrier in each coven. It has N/2 parties. Everytime a demon hits the main diagonal it awaits the demonsBarrier and goes to sleep. When N/2 demons await the demonsBarrier they wake up and start working again
- *Task 4*
  - Works exactly like the one presented before at Task 3 but the CyclicBarrier class is defined by me. The class has a lock for the counter, a counter and the number of parties. The constructor defines the numbes of parties. It has one method calles await. This method increases the counter and while the counter is smaller than the number of parties the demon is placed in an infinite loop that keeps him from working .

Observations

- The code I've sent in the arhive has most of the System outputs commented. I've kept only the ones related to undeads and witches. In order to check the functionality of the code each System output should be uncommented one by one. If I keep all of them the threads are very fast and the output is unreadable.
- In my opinion the times for threads should've been way bigger in order to be readable. For the sake of the problem statement I've kept them as required.
- The time between witch visits another coven is not defined in the problem statement. I've assumed it to be 10-30 ms.
- I've tried to use thread safe containers for data keeping like the ConcurrentHashMap.
- Added the gameEnded functionality when all undeads are defeated. This isn't presented on the problem statement.
- I've tried to keep the code as random as possible, even when we pick ingredients for potions.

Conclusions

- It's the first time when I work on a project this big which is Thread based. I've been into many problems caused by the synchronization methods. This way I've learnt more about the locks and what can happen if we forget to unlock it or any mistake like this. The code won't work at all. Java is not a language I've used a lot, this way I improved my Java skills.

References

- http://www.objectaid.com/
- https://stackoverflow.com/questions/35559129/why-does-clear-hashmap-method-clears-added-map-in-array-list
- https://stackoverflow.com/questions/10079266/copying-a-hashmap-in-java
- https://stackoverflow.com/questions/5516917/java-do-something-x-percent-of-the-time

- https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CyclicBarrier.html
- www.overleaf.com