Paradigm homework 2

Ting Yan

ID: 010765513

Design:

This homework is a producer/consumer multi-thread problem. In my weaponpart class, it has 4 subclass: steel, gunpowder, wood, rope. In my weapon class, it has 3 subclass: gun(made of steel, wood, gunpowder), cannon(made of steel, gunpowder, rope) and arrow(made of wood, steel and rope). So the producers are divided into 4 groups, each group makes a corresponding weapon part. Same as consumer, which are divided into 3 groups to make a specific type of weapon.

Implementation process:

Each castle has 14 workers, each worker is a thread. Castle A has 11 producers and 3 consumers. Castle B has 10 producers and 4 consumers. A producer that generates weapon part to be consumed by the consumer, however the rate of production and the rate of consumption vary.

There are 4 shared queues, steelQueue, woodQueue, gunpowderQueue and ropeQueue to co-ordinate all the workers. Here I used blocking Queue, since it is much easier and is itself thread safe, even though we have not implemented the synchronization ourselves.

Problems:

Thread synchronization and avoiding deadlock are the main issue we need to solve. At first, I tried to use Lock, which has better control over thread synchronization, but are more complicated to use. I google this topic and found many sample code for producer-consumer problems. But all the sample code are easy ones, they only have one shared queue or buffer. Each producer and consumer only work with one queue. But in our assignment, the consumer needs to use more than one queue, which is hard to coordinate. After reading the book "how to Program Java" Chapter26, I found that blocking queue is much easier and thread safe. So I chose to implement the thread using blocking queue.

Result:

Blocking queue works well for this project. Even though I finished this assignment and got a better understanding of multithreading, I still feel confused about how to use lock. I will continue to read and practice this topic, hope I can fully understand it. This homework is much harder than homework 1.

Test result:

1.Test whether the worker threads of each castle are working at the same time:
From the result below you can see rope is produced much more than other weapon part since it takes less time to produce, and workers are working concurrently. Once the weapon parts for a weapon are ready, consumer starts to make weapon.

Result:

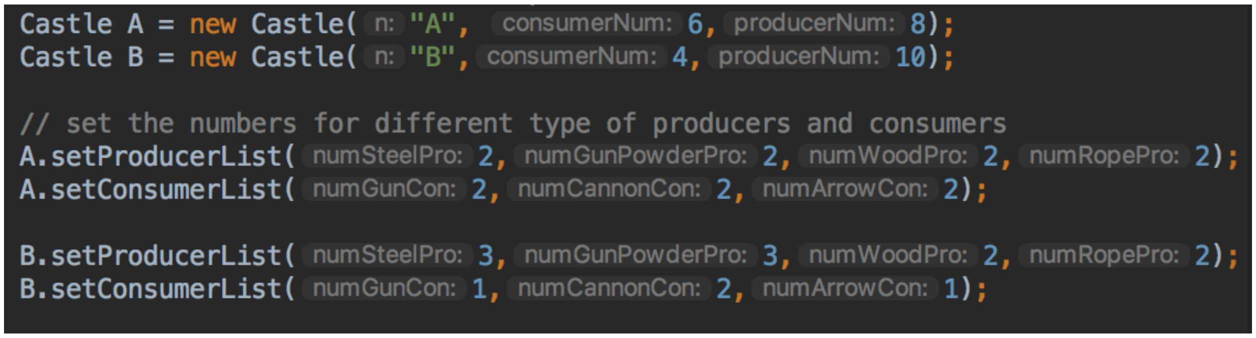
```
Producers of castle A start making weapon parts
Consumers of castle A start making weapon
Producers of castle B start making weapon parts
Consumers of castle B start making weapon
Castle B Producer#9 produced one rope

Castle B Producer#8 produced one rope

Castle A Producer#9 produced one rope

Castle A Producer#10 produced one rope

Castle A Producer#8 produced one wood

Castle B Producer#6 produced one wood

Castle A Producer#7 produced one wood

Castle B Producer#7 produced one wood

Castle A Producer#5 produced one gunpowder

Castle B Producer#4 produced one gunpowder

Castle B Producer#9 produced one rope

Castle A Producer#4 produced one gunpowder

Castle B Producer#8 produced one rope

Castle B Producer#5 produced one gunpowder

Castle A Producer#9 produced one rope

Castle B Producer#3 produced one gunpowder

Castle A Producer#6 produced one gunpowder

Castle A Producer#10 produced one rope

Castle A Producer#0 produced one steel

Castle A Consumer#0 Remove one steel and Wood and Gunpowder from queue
Castle A Consumer#0 produced one gun and add it to the armory.

Castle B Producer#2 produced one steel

Castle B Consumer#0 Remove one steel and Wood and Gunpowder from queue
Castle B Consumer#0 produced one gun and add it to the armory.

Castle B Producer#0 produced one steel

Castle B Consumer#1 Remove one steel and rope and gunpowder from queue
Castle B Consumer#1 produced one cannon and add it to the armory.

Castle B Producer#9 produced one rope

Castle A Producer#8 produced one wood

Castle B Producer#8 produced one rope

Castle A Producer#3 produced one steel

Castle A Consumer#2 Remove one steel and wood and rope from queue
Castle A Consumer#2 produced one arrow and add it to the armory.
```

2.Here is the result of the whole project (I have comment out the message from producer and consumer when they work, because that will make too many lines. I only left the message when castle gets attack and lost HP)

```
Producers of castle A start making weapon parts
Consumers of castle A start making weapon
Producers of castle B start making weapon parts
Consumers of castle B start making weapon
OUCH!!! Castle B lost 10 HP.
Current HP of castle B is 40

OUCH!!! Castle B lost 20 HP.
Current HP of castle B is 20

OUCH!!! Castle B lost 5 HP.
Current HP of castle B is 15

OUCH!!! Castle A lost 10 HP.
Current HP of castle A is 40

OUCH!!! Castle A lost 20 HP.
Current HP of castle A is 20

OUCH!!! Castle A lost 20 HP.
Current HP of castle A is 0

OUCH!!! Castle A lost 5 HP.
Current HP of castle A is -5
```

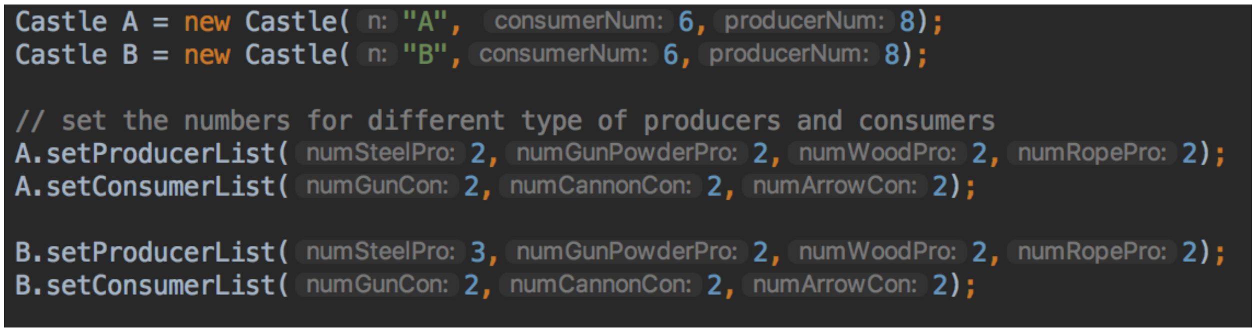
Castle A lose.
Castle A HP: -5
Castle B HP: 15

3. I tried to assign different numbers of workers to different tasks. It turns out the castle with more consumer will win.



```
Castle A = new Castle( n: "A",  consumerNum: 6, producerNum: 8);
Castle B = new Castle( n: "B", consumerNum: 4, producerNum: 10);

// set the numbers for different type of producers and consumers
A.setProducerList( numSteelPro: 2, numGunPowderPro: 2, numWoodPro: 2, numRopePro: 2);
A.setConsumerList( numGunCon: 2, numCannonCon: 2, numArrowCon: 2);

B.setProducerList( numSteelPro: 3, numGunPowderPro: 3, numWoodPro: 2, numRopePro: 2);
B.setConsumerList( numGunCon: 1, numCannonCon: 2, numArrowCon: 1);
```

in this situation, A wins



```
Castle A = new Castle( n: "A",  consumerNum: 6, producerNum: 8);
Castle B = new Castle( n: "B", consumerNum: 6, producerNum: 8);

// set the numbers for different type of producers and consumers
A.setProducerList( numSteelPro: 2, numGunPowderPro: 2, numWoodPro: 2, numRopePro: 2);
A.setConsumerList( numGunCon: 2, numCannonCon: 2, numArrowCon: 2);

B.setProducerList( numSteelPro: 3, numGunPowderPro: 2, numWoodPro: 2, numRopePro: 2);
B.setConsumerList( numGunCon: 2, numCannonCon: 2, numArrowCon: 2);
```

if A and B has same way of assigning works, it is a tie game.