CSC2002S August 2015

Assignment 2 – Golfer Game Report

Carla Wilby
WLBCAR002

Introduction

This report will serve to explain and motivate methodology used to create and ensure concurrency of this interpretation of the GolfGame assignment.

Class and methods brief:

DrivingRangeApp.java:
This is the main class of the golfGame. It takes in the command line parameters: Number of golfers, size of stash and size of bucket. It initializes the shared and global variables (array of golf balls, size of the array, size of golfer array, number of golfers), and threads – golfer threads and Bollie thread. It then runs these threads for between 10 and 20 seconds until the driving range "closes". At this time an AtomicBoolean 'done' is set to true so that the threads can see this and respond appropriately.
The DrivingRangeApp can be run in two modes – text based and visual based, which the user can choose.

Range.java:
This is the driving range simulation class. It contains an array the size of the ball stash (the maximum number of balls that can be on the field at any time) called ballsOnField, as well as the shared variables stash size and the atomic boolean cartOnField – which is set to true when Bollie is on the field. Since it is atomic, the value of this boolean can be seen and set in any of the necessary classes and reflect appropriately.
Methods:

- collectAllBallsFromField(): This method is used when Bollie is on the field "collecting" the balls. The balls are stored in the ballsOnField array in the positions corresponding to the ball's unique ID. This method puts all the balls on the field into new array the size of the amount of balls on the field, and sets the positions in the original ballsOnField array to null (representing no golf ball), and returns the new array.
- howManyBallsOnField(): This method counts the number of golfballs in the ballsOnField array.
- hitBallOntoField(): This method adds a specified golfball to the ballsOnField array, in the position corresponding to its unique ID number.
- lineToPrint(): This method is used in the visual version, it prints out a string representing the balls that are currently on the range.

BallStash.java:
This is the object representing the collection (an array is used) of golfballs in the stash – unused balls that golfers can add to their buckets. It has a number of synchronized methods for safely manipulating the stash.
Methods:

- enoughBalls(): This method calculates whether there are enough balls to fill a golfer's bucket left in the stash. This method is synchronized so that no bad interleavings can occur while the balls are being counted.

- getBucketBalls(): This method uses enoughBalls() to check that there are enough balls in the stash to fill a bucket. If there is, then the first available balls in the stash are taken out, put into a new array, and replaced with null. If there aren't enough balls, the method calls this.wait() on the current thread so that it is forced to wait until it is notified (this happens only when more balls are added to the stash).
- addBallsToStash(): This method takes the array of collected balls from Bollie and puts them into their corresponding positions (depending on their ID's) in the stash. It then calls the method golfersGo() which notifies all threads that are waiting for the stash to be refilled. This method is synchronized so that the stash cannot be changed while balls are being added and cause bad interleavings.
- golfersGo(): Notifies all waiting threads that the stash has been refilled.
- getBallsInStash(): Counts the number of balls currently in the stash, also synchronized for the same reason.

Golfer.java:
This object represents a golfer thread. It is initialized with the shared stash, field and Atomic boolean cartFlag variables, as well as a golferBucket (initially empty) and a new random swingtime for each golfer's swing and a semaphore, which will be used to halt all threads that are waiting for Bollie to leave the field. The run method sets a golfer thread into action. It runs as long as the Atomic boolean done does not reflect that the driving range is closing. Each golfer starts by filling their buckets of balls, and then hitting them off one by one, with a random swing time between 0 and 2 seconds. As they hit them off the balls are removed from their buckets and added to the range. If at any point Bollie comes onto the range then the golfers can finish their current swing (if applicable) and then must wait. This is done using a semaphore where all the permits are being held by Bollie, so they must wait at semaphore.acquire(). When Bollie releases the permits (when it is done) they quickly acquire and release the semaphore and continue to the next ball.

Bollie.java:
Bollie is the thread that represents the collector of balls from the range. Bollie is initialised with access to the shared stash and field object, as well as the Atomic done and cart on field (called bollieOnField here) flags, and the same semaphore that is passed into Golfer. Bollie runs in the background for the entire duration of the game, and is activated every 2 to 7 seconds. When he is activated, the cart on field flag is set to true, and the Golfer threads must finish their current threads and wait. This waiting is achieved as Bollie drains all the semaphore permits (so that there are none for the threads to acquire while he is on the field). Bollie sleeps for a set period of 1 second to simulate collection of balls, and then adds the balls collected from the field to the stash. Once this is complete the Atomic boolean cart on field is set to false and the semaphore permits are released so that the waiting Golfer threads can continue.

GolfBall.java:
This is the golf ball object. It is initialised as having a number between zero and the number of balls, in the order in which they are created by the driver class. Each ball can be tracked using this ID.

VisualGame.java:
Initializes the visual component of the game – this includes the drawing of golfers, bollie and the buckets. These methods are run from inside the VisualGolfer and VisualBollie classes.

VisualGolfer.java and VisualBollie.java
These objects are the same as the golfer and bollie classes except for minor changes in terms of the displaying of the results – done with the VisualGame class.

Concurrency requirements of the code:

Mutual exclusion:
In this program there was lots of room for data races and bad interleavings. Numerous golfer threads trying to access the same stash at the same time meant that much precaution needed to be taken to prevent data being lost, or the execution of the data under one thread influencing the outcome of another. In order to do this, all of the methods in the BallStash class were synchronized, so that only one thread could access the shared array at once.

Deadlock prevention:
Deadlock could have occurred when a golfer thread acquired a synchronized lock for one method, and requested the lock held by another thread. This was prevented by ensuring that only one lock was ever acquired at a time by any of the golfer threads, and that the golfer threads were forced to wait before Bollie could begin.

Java concurrency features used:

- Atomic boolean: These were used to specify the done flag and the Bollie on field flags. Since they are atomic, it is inherently synchronized, and a change made in one class or thread would be seen in the others. This made it a useful implementation for telling the threads that the
- Synchronized classes: This Java concurrency property was used on all methods where a thread could access the shared stash of balls object.
- Wait notify: This feature of Java locks was used when a golfer thread ran out of balls in its bucket, and there were insufficient balls in the stash to refill it. In this case, wait() was called to force the thread to wait until it is released. The releasing (notifyAll()) was run by Bollie after he had collected and added more balls to the stash.
- Semaphores: This form of Java concurrency was used as an alternative method to a conditional (spinning) while loop, as the threads wait for Bollie to leave the field. This involved forcing threads to wait to acquire a permit (held by Bollie), until he released the permits (see improvements and considerations for further clarification).

Validation of processes:

Since threads have a non-deterministic property, it is impossible to predict the output, and thus very difficult to run an effective testing mechanism. For this reason, validation for this assignment primarily involved manually testing the code in two ways:
1. multiple times with the same input, and then when that was working properly for all cases (deduced by manually examining for each test), then,
2. testing multiple different cases with different command line parameters and observing that it still abides by the correct methodology and sequence of output, returning no errors.

Using unusual parameters, like: 5 golfers, 20 balls in stash, 10 per bucket, proved to be a good way to ensure that the program was abiding by expectation (waiting correctly, filling correctly etc), even when the input was unreasonable for the code.

Improvements and considerations:

Some seemingly strange things where implemented in this code, so hopefully this will provide some clarification:

1. Array positioning: I used an array to represent both the stash and the balls on the field at any time. Since they are arrays, they needed to be statically assigned a size (that of the stash when it is full). Initially it made sense to me to then take advantage of such big arrays in carefully tracking the golf balls, so instead of placing an added ball anywhere in either of the

arrays, it is always placed in the position corresponding to its ID. Although this gives no real advantage (as I discovered as I got further with the code), it certainly made it easier to track specific balls and ensure that they were being correctly passed between arrays.

2. Using semaphores instead of spinning: My initial interpretation of the project included a spinning while loop so that the current running thread would not progress until bollie was off the field, it looked something like this:

```
        }
    while(cartOnField.get()==true){
        //do nothing
    }
```

Although this worked well for it's purpose, spinning is inherently a bad technique as it consumes a large amount of processing power while no progress is made. In order to counter this I wanted to use a Java synchronized technique to force the threads to wait. Unfortunately, I had already used wait() and notifyAll(), and I was afraid another implementation would cause threads to wait and be released in strange orders (in hindsight, there probably is a way to get around this, but using a completely different synchronization method made sense at the time, hence the birth of my semaphore!). The while loop was then replaced with an if statement as follows:

```
if(cartOnField.get()==true){ try {
    semaphore.acquire();                                    //Golfer tries to acquire a permit, must wait for bollie
      } catch (InterruptedException ex) {
          Logger.getLogger(Golfer.class.getName()).log(Level.SEVERE, null, ex);
      }
    semaphore.release();                                    //Release immediately after it is acquired
  }
}
```

This works when bollie holds all available permits, until he is finished, so the threads entering the if-statement are forced to wait at semaphore.acquire() until he is finished and releases the threads. The actual acquiring of the permits is redundant, so when he is finished, waiting threads acquire and then immediately release permits, and then are free to continue.

3. Visual component: this allows for the user to get a better idea of which balls are where at each time – in a specific golfer's bucket, on the field or in the stash (invisible).

Example output:

Text-based version: for input parameters: 3 Golfers, 20 balls, and 5 balls per bucket

```
=======   River Club Driving Range Open  ========
======= Golfers: 3 balls: 20 bucketSize: 5  ======
>>> Golfer #3 trying to fill bucket with 5 balls.
>>> Golfer #2 trying to fill bucket with 5 balls.
>>> Golfer #1 trying to fill bucket with 5 balls.
<<< Golfer #3 filled bucket with  5 balls (15 balls remaining in stash).
<<< Golfer #1 filled bucket with  5 balls (10 balls remaining in stash).
<<< Golfer #2 filled bucket with  5 balls (5 balls remaining in stash).
Golfer #3 hit ball #0 onto field
Golfer #2 hit ball #10 onto field
Golfer #1 hit ball #5 onto field
Golfer #1 hit ball #6 onto field
Golfer #3 hit ball #1 onto field
Golfer #2 hit ball #11 onto field
Golfer #3 hit ball #2 onto field
Golfer #3 hit ball #3 onto field
Golfer #1 hit ball #7 onto field
Golfer #2 hit ball #12 onto field
Golfer #1 hit ball #8 onto field
Golfer #3 hit ball #4 onto field
>>> Golfer #3 trying to fill bucket with 5 balls.
<<< Golfer #3 filled bucket with  5 balls (0 balls remaining in stash).
Golfer #2 hit ball #13 onto field
Golfer #3 hit ball #15 onto field
```

```
Golfer #1 hit ball #9 onto field
>>> Golfer #1 trying to fill bucket with 5 balls.
Golfer #3 hit ball #16 onto field
Golfer #3 hit ball #17 onto field
********** Bollie collecting balls   ************
********** Bollie collected 17 balls ************
Golfer #2 hit ball #14 onto field
********** Bollie adding 17 balls to stash, 17 balls in stash ************
<<< Golfer #1 filled bucket with  5 balls (12 balls remaining in stash).
>>> Golfer #2 trying to fill bucket with 5 balls.
<<< Golfer #2 filled bucket with  5 balls (7 balls remaining in stash).
Golfer #3 hit ball #18 onto field
Golfer #3 hit ball #19 onto field
>>> Golfer #3 trying to fill bucket with 5 balls.
<<< Golfer #3 filled bucket with  5 balls (2 balls remaining in stash).
Golfer #3 hit ball #10 onto field
Golfer #3 hit ball #11 onto field
Golfer #3 hit ball #12 onto field
Golfer #1 hit ball #0 onto field
Golfer #2 hit ball #5 onto field
********** Bollie collecting balls   ************
********** Bollie collected 8 balls ************
Golfer #3 hit ball #13 onto field
Golfer #1 hit ball #1 onto field
********** Bollie adding 8 balls to stash, 10 balls in stash ************
Golfer #3 hit ball #15 onto field
>>> Golfer #3 trying to fill bucket with 5 balls.
<<< Golfer #3 filled bucket with  5 balls (5 balls remaining in stash).
Golfer #2 hit ball #6 onto field
Golfer #1 hit ball #2 onto field
=======  River Club Driving Range Closing ========
Golfer #2 hit ball #7 onto field
Golfer #3 hit ball #0 onto field
Golfer #2 hit ball #8 onto field
Golfer #2 hit ball #9 onto field
>>> Golfer #2 returned with empty bucket.
Golfer #1 hit ball #3 onto field
Golfer #3 hit ball #5 onto field
Golfer #3 hit ball #10 onto field
Golfer #3 hit ball #11 onto field
Golfer #3 hit ball #12 onto field
>>> Golfer #3 returned with empty bucket.
Golfer #1 hit ball #4 onto field
>>> Golfer #1 returned with empty bucket.
BUILD SUCCESSFUL (total time: 18 seconds)
```

Visual version: for input parameters: 3 Golfers, 20 balls, and 5 balls per bucket

(considerable chunk cut off due to unprecedented length, run simulation manually to see full output.)

```
=======   River Club Driving Range Open  ========
======= Golfers: 3 balls: 20 bucketSize: 5  ======
********** Welcome to the visual golf game! **********
 _____            _____            _____
'     \\         '     \\         '     \\
    O//              O//              O//
    \_\              \_\              \_\
     | |              | |              | |
    /  |             /  |             /  |
   /   |            /   |            /   |
Golfer #1 Golfer #2 Golfer #3
|--|     |--|     |--|
|--|     |--|     |--|
|--|     |--|     |--|
+--+     +--+     +--+

Golfing range:********************
>>> Golfer #1 trying to fill bucket with 5 balls.
<<< Golfer #1 filled bucket with  5 balls (15 balls remaining in stash).
>>> Golfer #3 trying to fill bucket with 5 balls.
<<< Golfer #3 filled bucket with  5 balls (10 balls remaining in stash).
>>> Golfer #2 trying to fill bucket with 5 balls.
<<< Golfer #2 filled bucket with  5 balls (5 balls remaining in stash).
Golfer #2 hit ball #10 onto field
Golfer #1 hit ball #0 onto field


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #2
```

```
|  10|
| 11 12|
| 13 14|
+--+              0         10
Golfing range:********************


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #1
|  0|
| 1 2|
| 3 4|
+--+              0         10
Golfing range:********************
Golfer #1 hit ball #1 onto field
Golfer #3 hit ball #5 onto field


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #1
|  0|
| 1 2|
| 3 4|
+--+              01   5    10
Golfing range:********************


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #3
|  5|
| 6 7|
| 8 9|
+--+              01   5    10
Golfing range:********************
Golfer #2 hit ball #11 onto field


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #2
|  10|
| 11 12|
| 13 14|
+--+              01   5    1011
Golfing range:********************
Golfer #3 hit ball #6 onto field
Golfer #2 hit ball #12 onto field
Golfer #1 hit ball #2 onto field


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #3
|  5|
| 6 7|
| 8 9|
+--+              012  56   101112
Golfing range:********************


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #2
|  10|
| 11 12|
| 13 14|
+--+              012  56   101112
Golfing range:********************


 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #1
|  0|
| 1 2|
| 3 4|
+--+              012  56   101112
```

```
Golfing range:********************
Golfer #1 hit ball #3 onto field
Golfer #2 hit ball #13 onto field

 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #1
|  0|
| 1 2|
| 3 4|
+--+              0123 56   10111213
Golfing range:********************
Golfer #1 hit ball #4 onto field
Golfer #3 hit ball #7 onto field

 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #2
|  10|
| 11 12|
| 13 14|
+--+              01234567  10111213
Golfing range:********************

 _____
'     \\
    O//
    \_\
     | |
    /  |
Golfer #1
|  0|
| 1 2|
| 3 4|
+--+              01234567  10111213
Golfing range:********************
>>> Golfer #1 trying to fill bucket with 5 balls.
<<< Golfer #1 filled bucket with  5 balls (0 balls remaining in stash).
BOLLIE ON THE RANGE...

           ____|~\_
~~~~~~~~~[     _|_|-]
          (_)   (_)
********** Bollie collecting balls   ************
********** Bollie collected 14 balls ************
Golfer #2 hit ball #14 onto field
********** Bollie adding 14 balls to stash, 14 balls in stash ************
```

Conclusion

It works!

```
+        o       +                o
  +                  o     +           +
o             +
  o   +                 +         +
+           o     o        +         o
-_-_-_-_-_-_-_-_,------,      o
_-_-_-_-_-_-_-_-|    /\_/\
-_-_-_-_-_-_-_-~|__( ^ .^)   +    +
_-_-_-_-_-_-_-_ ""  ""
+        o        o         +        o
 +        +
o       o         o       o       +
  o           +
+       +       o       o       +       o
```